ARCADA

Utveckling av nivåvaktsystem för avloppstankar

Lucas Granberg

Examensarbete

Informations- och medieteknik

2013

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informations- och medieteknik
Identifikationsnummer:	4164
Författare:	Lucas Granberg
Arbetets namn:	Utveckling av nivåvaktsystem för avloppstankar
Handledare (Arcada):	Göran Pulkkis
Uppdragsgivare:	Smartel Electronics Oy

Abstrakt:

Uppdragsgivaren Smartel Electronics Oy har en produkt på marknaden som är en trådlös nivåvakt för avloppstankar. Produkten består av en inne- och ute-enhet. Ute-enheten kontrollerar nivån i tanken och skickar en trådlös signal till inneenheten. När Innenheten får ett larm börjar den blinka och pipa. Produkten är begränsad av att larmet bara kan ses på det ställe som inne-enheten befinner sig. Därför ville uppdragsgivaren utveckla produkten så att larmet kan skickas till en server över Internet och därifrån vidare till kunden via SMS eller e-post. Tjänsten Smartlarm utvecklades för detta ändamål. Smartlarm består av ny hårdvara som bygger på den gamla samt en server som tar hand om kommunikationen mellan nivåvakten och kunden. Examensarbetets målsättning är att dokumentera och motivera de val som görs samt de problem som uppstår i samband med utvecklingen av tjänsten. Syftet för examensarbetet är att det skall kunna användas som referens vid vidareutveckling av tjänsten. Tjänsten utvecklas genom ett nära samarbete med uppdragsgivaren eftersom både server och nivåvaktens hårdvara måste fungera tillsammans. Hårdvaran hör inte till detta examensarbete men sätter ändå vissa krav på hur serverlogiken utvecklas. Serverlogiken bygger på PHP-ramverket Symfony. Ramverket har fungerat som en röd tråd för hur projektet utformats eftersom andra projekt byggda med detta ramverk har använts som referens. Projektet ledde till en fungerande prototyp som kan visas för potentiella kunder. Examensarbetet fungerar som en bra referens för framtida utveckling av projektet.

Nyckelord:	Symfony, Doctrine, Smartel Electronics Oy, nivåmätning, avloppstank, produktutveckling, prototyp, dokumentering
Sidantal:	53
Språk:	Svenska
Datum för godkännande:	19.2.2012

DEGREE THESIS	
Arcada	
Degree Programme:	Information and Media Technology
Identification number:	6164
Author:	Lucas Granberg
Title:	Development of sewer tank level alarm
Supervisor (Arcada):	Göran Pulkkis
Commissioned by:	Smartel Electronics Ltd.

Abstract:

Smartel Electronics Ltd. has a product on the market that is a wireless float switch alarm for sewer tanks. The product consists of an indoor and an outdoor unit. The outdoor unit checks the level in the tank and sends a wireless signal to the indoor unit. When the indoor unit receives the alarm it starts flashing and beeping. The product is limited by the fact that the alarm is only visible at the location of the indoor unit. Therefore the commissioner of this thesis wanted to develop the product so that the alarm can be sent to a server over the Internet and from there on to the customer via SMS or email. The Smartlarm service was developed for this purpose. Smartlarm consists of new hardware based on the old and a server that takes care of the communication between the hardware and the client. The thesis aims to document and justify the choices made and the problems that arise in connection with the development of the service. The purpose of the thesis is that it can be used as reference for further development of the service. The service is developed through a close collaboration with the commissioner as both the server and the level switch hardware must work together. The hardware does not belong to this thesis but nevertheless puts certain requirements on how the server logic is developed. Server logic is based on the PHP framework Symfony. The framework has functioned as a guideline for development of the project because other projects built with this framework have been used as a reference. The project produced a working prototype that can be shown to potential customers. The thesis serves as a good reference for future development of the project.

Keywords:	Symfony, Doctrine, Smartel Electronics Ltd., level measurement, sewer tank, product development, prototype, documentation
Number of pages:	53
Language:	Swedish
Date of acceptance:	2013-02-19

INNEHÅLL

FIGURER

FÖRKORTNINGAR OCH BEGREPP

ACL Access Control List. Ett system för att hålla reda på användarrättigheter för data. Ursprungligen utvecklat för filsystem men samma tankesätt kan implementeras för t.ex databasresurser.

ADC Analog-to-digital converter. På svenska kallad A/D-omvandlare. En krets som tar in en analog signal och omvandlar den till ett digitalt format.

Adobe Flash En multimedia- och mjukvaruplattform. Kan användas som ett tillämpningsprogram som körs i en webbläsare men kräver då att tilläggsmoduler installeras i webbläsaren. Flash ägs av företaget Adobe.

Bitbucket.com En webbaserad hostingtjänst lik Github.com. Stöder både GIT och Mercurial.

Bootstrap En samling kod och verktyg för att skapa användargränssnitt på webben. Bygger på HTML, CSS och JavaScript. Gjort av företaget Twitter Inc.

BootstrapCDN En gratis CDN för ramverken Bootstrap och Font Awesome. Upprätthållen av CDN-tjänsteleverantören MaxCDN.

CDN Content Delivery Network. Ett system av servrar för att försnabba nerladdning och distribution av stora filer över internet.

CMS Content Management System. En samling verktyg samlade under samma portal för att producera och manipulera data på en webbsida.

CRC Cyclic Redundancy Check. Ett sätt att räkna kontrollkoder för data. Kan användas för att kontrollera om data har överförts rätt eller om data har ändrat genom att kontrollera kontrollkoder.

CSS Cascading Style Sheets. En fil som innehåller information om hur data skall presenteras. På svenska även kallad stilmall.

CSV Comma-Separated Values. Ett filformat för att spara datatabeller. Använder kommatecken för att separera datakolumner och ny rad för att separera rader.

DBAL DataBase Abstraction Layer. Ett abstraherat objektorienterat gränssnitt till databasen. DBAL möjliggör att olika system med samma kod kan använda olika databaser.

Drupal 8 Ett CMS under utveckling som bygger på Symfony-komponenter.

Facebook En social nätverkstjänst med över en miljard användare. Är också en oAuth-tjänsteleverantör.

Flotcharts Ett JavaScript-bibliotek för att visualisera data på samma sätt som Google Chart Tools. Bygger på jQuery.

GIT Ett versionshanteringssystem för programvaruprojekt. Används mycket av PHP- och Symfony-användare.

Github.com En webbaserad hostingtjänst för GIT-baserade projekt. Används mycket av öppen källkod-entusiaster. Innehåller verktyg som underlättar samarbete och sammanvändning av kod.

Google Chart Tools En tjänst gjord av Google som låter användare generera visuella representationer av data såsom stapeldiagram och pajdiagram.

GPRS General Packet Radio Services. En paketbaserad tjänst för överföring av data i GSM-nätverk.

GSM Ursprungligen stod GSM för Groupe Spécial Mobile men står på svenska för Globalt system för mobil kommunikation. GSM möjliggör trådlös telefon- och SMS-kommunikation.

HTML Hypertext Markup Language. Ett märkspråk som används för att bygga upp de flesta webbsidor.

HTTP HyperText Transfer Protocol. Ett applikationsprotokoll för webkommunikation.

ICCID En unik nummerkod för SIM-kortet. Inte samma sak som telefonnumret.

IDE Integrated Development Environment. På svenska integrerad utvecklingsmiljö. Ett datorprogram med diverse verktyg avsedda för att underlätta programvaruutveckling.

IMEI International Mobile Equipment Identity. En unik nummerkod för GSM-enheter.

JavaScript Ett skriptspråk som främst körs i webbläsaren för att skapa dynamiskt innehåll på en webbplats. Stöd för JavaScript finns i alla moderna webbläsare men koden kan också köras på en server.

jQuery Ett JavaScript-kodbibliotek för att underlätta utvecklingen av JavaScript-komponenter.

JSON JavaScript Object Notation. En öppen standard för ett textbaserat dataformat.

Mercurial Ett versionshanteringssystem för programvaruprojekt. Kan jämföras med och är mycket likt GIT.

MPLABX En IDE gjord av Microchip för deras programmerbara integrerade kretsar. MPLABX bygger på Netbeans IDE. Innehåller förutom Netbeans-funktionalitet även stöd för kompilatorer och avlusare (engelska debugger). MPLABX har också stöd för programmeringshårdvara för att föra över kompilerad kod till programmerbara integrerade kretsar.

MVC Model-View-Controller. Ett mönster för mjukvaruarkitektur.

Netbeans En IDE för en mängd olika programmeringsspråk såsom t.ex. PHP, HTML5 och JavaScript.

oAuth En standard för autentisering på webben. Genom att ha ett användarkonto i en tjänst som erbjuder oAuth kan andra tjänster låta dessa logga in med sina färdiga konton och behöver således inte registrera sig på nytt.

Oracle Java Java är ett programmeringsspråk som ägs av Oracle. Med Java kan man precis som med Adobe Flash göra tillämpningsprogram som körs i webbläsaren. Java kräver också att tilläggsmoduler laddas ner för att kunna köras.

ORM Object-Relational-Mapping. Datan i databasen kopplas ihop till objekt i programmet med hjälp av en DBAL.

PhpUnit Ett testramverk för PHP-kod. Innehåller funktionalitet för att automatisera körningen av test som skrivs av utvecklaren.

Projektet Om ingenting annat nämns avses med denna term hela produktutvecklingsprocessen inklusive hårdvaruutveckling.

REST REST står för REpresentional State Transfer. En arkitektur och ett tankesätt för hur man skall utveckla nätbaserade tjänster.

SIM Subscriber Identity Module. En krets avsedd att användas med mobila GSM-enheter, utfärdat av en mobiloperatör. Kretsen innehåller information om abonnemanget samt krypteringsfunktioner för datatrafiken. Distribueras oftast i form av ett plastkort som sätts in i t.ex. en mobiltelefon. Kan också vara en integrerad krets fastlödd i ett kretskort.

Sim900 Sim900 är en hårdvarumodul som tar hand om GPRS-trafiken och kontakten med SIM-kortet. Mer om Sim900 under dess dedikerade avsnitt.

Smartlarm-klient Själva hårdvaran som består av mätutrustning, mikrokontroller och Sim900 modul.

SMS Short Message Service. En tjänst för korta textmeddelanden som skickas över GSM-nätverk. SMS-meddelanden skickas främst mellan mobiltelefoner.

SOAP Simple Object Access Protocol. Ett XML-baserat protokoll för nätbaserade tjänster.

Stackoverflow En webbportal för svar och frågor som berör det mesta som har med programmering att göra.

Symfony Symfony är det PHP-ramverk som används på servern.

Symfony2 Symfony är hela tiden under utveckling men det mesta i ramverket gjordes om från början för version två. Symfony2 är namnet på denna version. I texten nedan avses Symfony2 när namnet Symfony nämns.

UNIX Ett operativsystem utvecklat av företaget AT&T Bell Labs i USA under 1960- och 1970-talet.

WDT WatchDog Timer. En krets i en integrerad krets som används bland annat för att väcka upp kretsen ur strömsparläge. Består av en klocka som räknar upp eller ner och när klockans värde når ett förutbestämt värde skickas en signal vidare till en WDT-styrd funktion. Kan också användas för att starta om datorsystem efter en viss tid. Användbart som en sista utväg när datorprogram låser sig.

Wrapbootstrap En webbplats som säljer mallar för Bootstrap med CSS, JavaScript och HTML. Mallarna består av HTML -filer som beskriver hur man skall använda mallen och demonstrerar de stilsatta element som mallen innehåller såsom t.ex. tabeller, menyer och grafer.

XML Extensible Markup Language. Ett utbyggbart märkspråk som används för att formatera data.

YAML YAML är en rekursiv akronym som står för YAML Ain't Markup Language. Ett sätt att formatera data. Används främst i konfigurationsfiler.

1 INLEDNING

Det här examensarbetet bygger på ett projekt för Smartel Oy som fungerar som uppdragsgivare. Examensarbetets målsättning är att dokumentera och motivera de val som gjordes samt de problem som uppstod i samband med utvecklingen av projektet. Syftet för examensarbetet är att det skall kunna användas som referens vid vidare utveckling av projektet.

1.1 Utgångspunkt

Smartel Electronics Oy sysslar med småskalig produktion av elektronik. Företaget har en produkt på marknaden i form av en trådlös nivåvakt för avloppstankar. Den består av en sändare och en mottagare.

Sändaren mäter nivån i en avloppstank och skickar ett alarm till mottagaren när vätskenivån går över eller under ett förutbestämt värde. Sändaren är batteridriven med en beräknad livslängd på 10 år. Den är spolsäker för att motstå fukt och eländiga förhållanden. Mottagaren tar emot alarmsignalen och varnar användaren med hjälp av ljud- och ljussignaler.

Eftersom alarmet endast är synligt på ett ställe har behovet uppstått att få det skickat via SMS eller e-mail till användaren. Den nya produkten skall ha minst lika bra nivåmätningsegenskaper, livslängd samt hållbarhetsgrad i finska utomhusförhållanden som den gamla produkten.

För att kunna kontakta kunden via andra elektroniska medel krävs det en del ändringar gällande hur hela produkten fungerar. Mottagaren faller bort från produkten och ersätts med en server som tar emot alarm via GPRS. Det här examensarbetet bygger på utvecklingen av det nya larmsystemet.

Larmsystemet består av en klient och en server. Klienten innehåller en mikrokontroller, en sensor, en GPRS-modul, ett batteri samt elektronik och mjukvara som kopplar samman dessa komponenter. Servern består av ett klientgränssnitt, ett användargränssnitt samt ett administratorgränssnitt.

1.2 Målsättningar

Målsättningen med uppdragsgivarens projekt är att utveckla ett komplett produktpaket som kan säljas som en tjänst till privatpersoner och företag. Målsättningen med detta examensarbete är att definiera, utveckla och dokumentera denna produkt.

Vid projektets slut skall en fungerande prototyp vara redo för produktion. Klienten skall klara av att göra sin uppgift och kommunicera med servern. Servern skall ha definierade gränssnitt och krav för dessa. Dessa krav skall uppfyllas och utvecklingsarbetet skall dokumenteras.

1.3 Avgränsningar

Eftersom examensarbetet är gjort inom studieinriktningen informationsteknik tas inte klientens elektronik och annan hårdvara upp i detalj. Utvecklingen av klientens kod behandlas inte heller i detta examensarbete. Endast i de fall där krav på klienten, som t.ex. strömsnålhet, påverkar hur servern kommunicerar med klienten tas dessa aspekter upp.

Serverns hårdvara tas inte upp eftersom dess kod körs på en molntjänst. Hur kommunikationen sker i praktiken med GPRS nämns inte heller. Orsaken är att vi använder oss av färdiga GPRS-moduler.

2 TEKNOLOGI FÖR PROGRAMVARUUTVECKLING

Det finns många sätt att utföra arbetet i ett projekt som detta. Det finns ett flertal olika kodspråk, ramverk och tankesätt man kan välja mellan. Med olika kombinationer av dessa kan man åstadkomma samma sak. Det är ändå viktigt att välja den rätta teknologin som passar projektet och de som arbetar med det för att minimera kostnaderna.

I detta kapitel dokumenteras och motiveras de val som har gjorts från en teknologisk synvinkel. De flesta valen utgår från tidigare erfarenheter men de har också påverkats av trender och utveckling inom området.

En röd tråd som löper igenom alla beslut som har med projektet att göra är att använda färdig kod och färdiga lösningar så långt som möjligt. Många problem i detta projekt har andra utvecklare redan varit tvungna att lösa. Dessa lösningar kommer i form av olika ramverk och kodsamlingar. Det här examensarbetet dokumenterar inte färdig kod utan endast hur denna kod har blivit använd och till vilket ändamål.

2.1 Programmeringsspråk

Ett av de största och första besluten i ett utvecklingsprojekt för programvara är valet av programmeringsspråk. En bra minnesregel när det kommer till kodspråk är att det inte finns något perfekt kodspråk (Wikibooks, 2013). Med det i tankarna när man väljer kodspråk är det mest en

fråga om att hitta någonting som passar in i den miljö man valt att utveckla i och som passar ihop med alla andra komponenter man valt. Det här projektet består såsom tidigare nämnts av en klient och en server.

2.1.1 Klientens kod

Klientens kod hör inte till det här examensarbetet men det kan ändå nämnas att XC8 valdes. Mikrokontrollern är vald av uppdragsgivaren och den är producerad av Microchip. XC8 är i själva verket en kompilator för Microchips 8-bit-mikrokontrollers. Programmeringsspråket är en specialiserad version av C som följer ANSI-standarden, även kallad ANSI C.

Programmeringsspråket C utvecklades i början av 1970 parallellt med operativsystemet UNIX. Många programmeringsspråk bygger på eller är starkt influerade av C (Ritchie, 1993). Kompilatorer för C kan skrivas för så gott som alla plattformar. Detta ledde i början till att program skrivna för en kompilator inte kunde kompileras för en annan. Därför gjorde ANSI en standard för kodspråket. ANSI står för American National Standard Institute, en organisation som arbetar med standarder och normer (American National Standards Institute, 2013).

2.1.2 Serverns kod

Serverns kod består både av den kod som körs på servern och den som körs på användarens dator, även kallad för klientkod (inte att förväxla med klientens, d.v.s. hårdvarans kod som nämndes i förra stycket). Eftersom vi inte vill att användaren av sidan måste ladda ner tillägg till sin webbläsare såsom Adobe Flash eller Oracles Java valdes kodspråk som stöds av majoriteten av webbläsare. Dessa språk är HTML, CSS och JavaScript. HTML är ett märkspråk och innehåller strukturen och materialet som användaren ser i sin webbläsare. CSS används för att formatera och arrangera materialet som ett tillägg till HTML. JavaScript låter en göra dynamiskt innehåll på webbsidan utan att ladda ett helt nytt HTML-dokument varje gång man vill ändra något på sidan. Mer om detta finns i kapitel 3.1.

För serverns programmeringsspråk valde jag PHP. Valet gjordes av den enkla anledningen av att det är ett språk jag behärskar och för att det är lämpligt för projektet. Eftersom jag har arbetat

mycket med PHP vet jag till stor del hur man gör det som krävs för ett projekt som detta. Jag har också valt ramverket Symfony som är ett PHP-ramverk så valet var ganska självklart.

PHP är en så kallad rekursiv akronym som står för PHP: Hypertext Preprocessor. Den första versionen av PHP gavs ut år 1995 av Rasmus Lerdorf (The PHP Group, 2013). Därefter har PHP gått igenom ett flertal versioner och är nu uppe i PHP-version 5. PHP-kod tolkas på servern och tar in data från användare, databaser, filer och andra källor. Från att ha varit ett hobbyprojekt gjort för att räkna besökare på författarens hemsida har PHP vuxit till att i dagens läge användas på miljontals servrar runtom i världen.

En av de största nackdelarna med PHP är att det är enkelt att skriva dålig kod. Med dålig kod menar man kod som är svår att förstå sig på och som är svår att göra felsökning på. Detta är en sanning med så gott som alla kodspråk men framstår väldigt tydligt i PHP eftersom det har sitt ursprung i ett litet hobbyprojekt.

Nuförtiden är PHP ett välutvecklat kodspråk och det är lättare att skriva bra kod som är lätt att förstå sig på. Om man skriver mycket kod för ett komplext projekt som detta räcker det inte bara med ett bra kodspråk. För att skriva bra kod behöver man då också ett ramverk.

Det finns såklart andra programmeringsspråk såsom Python, Ruby och Java som också kunde användas för ett projekt som detta. T.ex. Python har många fina argument om varför just deras programmeringsspråk är bättre och lättare att förstå (Python Software Foundation, 2012).

2.1.3 Dokumentering

Oberoende vilket programmeringsspråk som används är det alltid en god idé att på något sätt dokumentera sin kod. Även om det just nu finns endast en utvecklare för projektet är det viktigt med god dokumentation för framtida utveckling eller för nya utvecklare i projektet (Moxcey, 2007). Även om fördelarna med att dokumentera sin kod är rätt så klara finns det också nackdelar. Problem uppstår om man i dokumentationen beskriver hur en metod arbetar. Extremerna är att antingen är koden självförklarande vilket betyder att dokumentationen inte ger något mervärde, eller så är dokumentationen fel vilket bara förvärrar situationen (Vogel, 2013).

För att undvika detta ges endast en kort beskrivning på alla metoder och klasser samt parametrar som matas in och resultat som returneras. På så sätt är man tvungen att skriva kod som är lätt att förstå samtidigt som man har in- och ut-variabler bestämda i dokumentationen.

För att dokumentera koden används PHP Documentor, även kallad PHPDoc. PHPDoc är ett verktyg som genererar dokumentation utgående från PHP-kod och kommentarer. Orsaken varför PHPDoc valdes framom andra liknande verktyg är:

- Ramverket Symfony (se kapitel 2.2) använder PHPDoc.
- Dokumentationen kan skrivas som kommentarer i koden vilket underlättar när man läser koden och när man skriver eller ändrar på koden. Allting finns på samma ställe och dokumentationen uppdateras till stora delar automatiskt.
- Netbeans IDE har stöd för PHPDoc. T.ex. när man skriver början på en metods namn kan Netbeans IDE ge förslag på resten och påminna en om vilka variabler som krävs och vad som returneras.

I det här skedet har dokumentationen inte genererats eftersom behovet inte har funnits. Trots detta är koden färdigt dokumenterad med PHPDoc-taggar vilket möjliggör att dokumentation genereras vid behov. Ett exempel på PHPDoc-syntax kan ses nedan (Teutsch, 2012):

```
<?php
/**
 * Calculates sum of squares of an array
 *
 * Loops over each element in the array, squares it, and adds it to
 * total. Returns total.
 *
 * This function can also be implemented using array_reduce();
 *
 * @param array $arr
 * @return int
 * @throws Exception If element in array is not an integer
 */
function sumOfSquares($arr) {
    $total = 0;
    foreach ($arr as $val) {
        if (!is_int($val)) {
            throw new Exception("Element is not an integer!");
        }
        $total += $val * $val;
    }
    return $total;
}
```

Hela kommentaren som börjar med “/*” och slutar med “*/” är ett så kallat DocBlock. Den här kommentaren har en viss struktur som PHPDoc kan läsa. Första raden är en kort förklaring på det som dokumenteras. Resten av texten är en längre förklaring med mer detaljerad information. Rader som börjar med ”@” är nyckelord som innehåller metadata. I det här fallet används nyckelorden ”@params”, ”@return” och ”@throws”. ”@Params” innehåller information om metodens parametrar namn och datatyp. Nyckelordet ”@return” berättar vilken typ av objekt metoden förväntas returnera. Nyckelordet ”@throws” förklarar vilka fel metoden kan stöta på och ge upphov till så kallade undantagstillstånd.

Även om PHPDoc är endast ämnat för PHP kan samma syntax användas av andra kodspråk också som t.ex jsDoc för JavaScript. Andra koddokumenteringsverktyg för PHP använder till stora delar samma syntax vilket möjliggör byte av dokumentationsgenerator.

2.2 Ramverket Symfony

Oberoende av programmeringsspråkets typ lönar det sig att bygga en applikation med hjälp av ett ramverk (Phpandstuff, 2009). Ett bra ramverk är en samling regler, tankesätt, strukturer och kod som hjälper en att lyckas med sitt projekt. Genom att använda ett ramverk kan man utveckla applikationen snabbare, bättre och säkrare.

- Utvecklingen blir snabbare p.g.a. att det finns färdiga kodfragment som man kan plocka in för de vanligaste uppgifterna såsom formulär, databaskoppling, layout osv.
- Utvecklingen blir bättre dels för att de används av många och dels för att dessa kodfragment och det egna programmet följer de regler och rekommendationer som ramverket lägger fram.
- Utvecklingen blir säkrare p.g.a. att det finns många som använder dessa kodfragment och testar dem.

Symfony är alltså ett ramverk som består av flera komponenter. En bra sak med Symfony är att alla dessa komponenter, eller bundles som de kallas i Symfonys terminologi, kan användas också utanför Symfonyramverket. T.ex. använder Drupal8, som är ett CMS under utveckling, delar av Symfony.

I ramverk förekommer termen MVC ofta. MVC är ett designmönster och står för Model View Controller. I korta drag kan man säga att Model-delen är kontakten till data. View-delen är det man ser, dvs. själva layouten och Controller är den del som tar emot förfrågningar av användaren, tar data från databasen via Model-delen och skickar tillbaks View-delen fylld med data. Idén med designmönstret MVC är att separera logikkod, layoutkod och databaskod.

Symfony följer MVC-strukturen men enligt Fabien Potencier, grundaren av Symfony, är det inte en särdeles lämplig term för ramverket Symfony(Potencier, 2001). Symfony har för det första ingen egen Model-del utan bara View- och Controller-delar. Model-delen kan skötas "manuellt" genom att skriva egen databaslogik eller genom att använda en färdig ORM (Object Relation Mapper) såsom Doctrine. Mer om detta finns i underkapitel 2.4. I grund och botten är Symfony ett HTTP-ramverk som tar emot förfrågningar och skickar tillbaks svar vilket är precis det som projektet kräver.

Jag valde Symfony av samma orsak jag valde PHP, nämligen att det är bekant från tidigare. Jag vet att det som krävs för projektet är genomförbart med Symfony. De delar som projektet krävde som inte fanns färdigt inbyggt i Symfony fanns istället som tilläggskomponenter.

2.2.1 Tilläggskomponenter

Såsom tidigare nämnts är hela ramverket Symfony uppbyggt av tilläggskomponenter eller s.k bundles. En bundle är alltså en fristående komponent som passar ihop med andra fristående komponenter som följer samma struktur. I det här kapitlet beskrivs några av de viktigaste komponenterna som valts till projektet.

Sonata AdminBundle är ett administratorgränssnitt som gör det möjligt för en användare med administratorrättigheter att söka, visa, lista och modifiera data. Det grafiska gränssnittet bygger på Bootstrap.

Sonata UserBundle gör en färdig användarentitet med de vanligaste fälten såsom användarnamn, lösenord och e-mail. Genom att använda en färdig struktur för användaren blir användarobjektet kompatibelt med andra tilläggskomponenter. T.ex. om ett behov uppstår att göra det möjligt för användare att logga in med sina Facebook-användarkonton via oAuth går det enkelt att göra.

Knplabs/Knp-Menu-Bundle gör det lättare att konstruera menyer. Varje länk i menyn kan tilldelas olika användarrättigheter så att menyn ser olika ut beroende på vem användaren är. Knp-Menu-Bundle passar också ihop med bootstrap.

Här har jag bara beskrivit de viktigaste komponenterna jag använt. För utförlig beskrivning se Bilaga 2.

2.2.2 Composer

Att hålla reda på alla tilläggsmoduler, deras versioner samt beroenden mellan dessa är väldigt tidskrävande. För detta ändamål används en PHP-applikation som heter Composer. Composer består bara av en fil, composer.phar. Phar står för PHP-Archive och är en fullständig PHP-applikation.

Allt som har att göra med Symfony kan man ladda ner via Composer. När man startar ett nytt Symfony-projekt är det enklaste sättet att först ladda ner Composer och sedan ladda ner Symfony via den.

Alla tilläggsmoduler skrivs in i filen composer.json som är en JSON-formaterad lista över vilka moduler man vill ladda ner samt deras versionsnummer (Bilaga 2). Alla tilläggsmoduler som kan laddas ner via Composer finns på webbplatsen packagist.org (Packagist, 2013). Ibland vill man ladda ner kod som inte finns med på packagist, t.ex. finns Twitters Bootstrap-kodbas uppehålls på Github. Då kan man istället skriva in GIT-adressen tillsammans med ett versionsnummer för att kunna ladda ner just den version man vill ha.

Genom att köra kommandot ”phpcomposer.phar update” laddar man ner alla filer man inte ännu har eller som har uppdaterats sedan man senast körde kommandot. Om versionskonflikter uppstår ger Composer en varning och beskriver vilka moduler som ger upphov till dessa konflikter. Oftast handlar det om att två moduler kräver olika versioner av en tredje modul. Dessa konflikter uppstår ofta när själva Symfony uppdateras och vissa moduler inte hänger med i utvecklingen.

Composer är alltså en enkel och användbar applikation som gör det lättare att utveckla i Symfony. Sökgränssnittet på sidan Packagist är också väldigt enkelt att använda för att hitta det man vill ha. Hittar man någonting som ser intressant ut är det enkelt att installera. Vill man ta bort

tilläggsmoduler ur projektet behöver man bara ta bort raden med den aktuella tilläggsmodulen från composer.json-filen. Därefter kör man uppdateringskommandot som beskrevs tidigare.

Genom att föra ett register över använda tilläggsmoduler är det lätt att flytta projektet från t.ex. en utvecklingsserver till en produktionsserver. Bara den kod som är skriven specifikt för projektet flyttas över och sedan laddas tilläggsmodulerna ner via Composer. Detta håller filöverföringen mellan servrarna liten och stora kodbibliotek kan laddas ner från snabbare servrar. Mer om detta finns i underkapitel 2.6.

2.3 Användargränssnitt

Servern har två användargränssnitt, ett administratorgränssnitt och ett kundgränssnitt. Dessa gränssnitt är tillgängliga via användarens webbläsare. Via dessa gränssnitt kan användaren söka i data, visa data och mata in ny data samt redigera gammal data.

För att kunna göra detta effektivt krävs ett användarvänligt gränssnitt. Det finns många åsikter om vad användarvänlighet på webben är. Med stöd av Jakob Nielsens teorier om användarvänlighet (Nielsen, 1995) och samtal med uppdragsgivaren har dessa krav ställts på användargränssnittet:

- Enhetlig layout och enhetliga färgteman så att användaren inte belastas med att lära sig flera olika utseenden.
- Kundgränssnittet skall fungera lika bra på såväl pekplattor och mobiltelefoner som stationära datorer.
- Ingen onödig information som skapar förvirring. Information som kan vara nyttig men som inte måste synas hela tiden ska gömmas undan.
- Texter på användarens språk. Finska och svenska är minimikrav medan engelska är en del av framtida utveckling.
- Nyttiga felmeddelanden så att användaren förstår när någonting gått fel och hur det skall åtgärdas t.ex. vid inmatning av data.
- Användargränssnitten skall inte behöva en separat dokumentation för att kunna användas.
- Användargränssnitten måste ladda snabbt och serverbelastningen måste minimeras.

Återigen följs idén med att inte uppfinna hjulet på nytt genom att använda färdiga ramverk och kodbibliotek där det är möjligt.

2.3.1 Bootstrap

För att uppnå tidigare nämnda krav på ett effektivt sätt används ett färdigt ramverk. Ramverket som valdes heter Bootstrap och är skrivet av Twitter. Bootstrap är det mest populära projektet på kodförvaringsplatsen Github. Popularitet tyder inte alltid på att det är bäst passande för ändamålet men ger ändå en god indikation på att detta är ett välskrivet och välunderhållet kodbibliotek. Det är också en god indikation på att det är den mest använda kodsamlingen på webben.

Ramverket Bootstrap valdes eftersom det är gratis, lättanvänt, vida använt inom Symfonys användarkrets och eftersom det är bekant från tidigare. Bootstrap används också som användargränssnittsramverk för administratorgränssnittet.

Genom att använda ett färdigt ramverk uppnår man ett enhetligt utseende och en enhetlig funktionalitet överallt i gränssnittet. Element såsom knappar, formulär och menyer följer alla samma utseende och ger på så vis ett enhetligt och lättnavigerat utseende. Det finns också färdiga teman för Bootstrap som hjälper en med val av färger och grundlayout.

För att användargränssnittet skall fungera lika bra på mobila enheter som på stationära datorer finns det två alternativ. Antingen bör man göra två olika versioner av användargränssnitten eller så gör man layouten responsiv. Eftersom det är besvärligt att upprätthålla flera versioner av samma gränssnitt valdes det senare alternativet. En responsiv layout anpassar sig till den skärm där den visas (Pettit, 2012).

Bootstrap har färdigt inbyggt stöd för responsiv design. När skärmen som sidan visas på blir mindre ändras materialet dynamistkt. Några funktioner som kan nämnas är:

- Bilder och text blir mindre.
- Skrymmande menyer fälls ihop.
- Rullgardinsmenyer som förut fungerade genom att sväva med musen över dem byts ut mot rullgardinsmenyer som fälls ut när man trycker på dem. Orsaken till detta är att pekskärmar inte stöder svävning med muspekare eftersom all interaktion tolkas som mustryckningar.

- En del material omstruktureras så att det viktigaste kommer först medan mindre viktigt material kan gömmas eller fällas ihop.

2.3.2 Font Awesome

För att göra användargränssnittet ännu mer lättanvänt och tydligt används olika ikoner. T.ex. i menyn används ikoner som symboliserar vad de olika menyalternativen står för. Dessa ikoner kommer från Font Awesome. Font Awesome är ett typsnitt med ikoner istället för bokstäver. Nyttan med att ha ikonerna inbakade i ett typsnitt är:

- Webbläsaren behöver bara ladda ner en typsnittfil istället för många bildfiler med ikoner i. Detta minimerar antalet förfrågningar till webbläsaren.
- Ikonerna är sparade som kurvor och inte som bilder vilket gör filstorleken mindre och ikonerna tappar inte kvalitet när man ändrar deras storlekar.
- Ikonernas storlek och färg kan ställas med CSS-stilar och -klasser.

Font Awesomes ikoner är stilrena och passar bra in med det övriga intrycket på sidan. Eftersom många andra sidor använder samma ikoner är det lätt att känna igen dem. Dessutom följer de bekanta normer. T.ex. representerar kugghjul inställningar och ett hus betyder hem eller framsidan.

Font Awesome kommer också med färdiga CSS-klasser för att göra ikonerna lätta att använda. Genom att t.ex. skriva ”`<i class="icon-spinner icon-spin icon-large icon-4x"></i>`” får man en stor ikon som representerar att någonting laddar. Ikonerna är gjorda för Bootstrap och kan laddas direkt från BootstrapCDN. Mer om detta finns i underkapitel 3.1.3.

2.4 Databasabstraktionsbiblioteket Doctrine

Eftersom det inte i skrivande stund är bestämt vilken sorts hårdvara och operativsystem servern skall köras på är det viktigt att inte låsa sig till en viss sorts databas. För att kunna lämna detta beslut till ett senare skede behövs ett abstraktionslager mellan databasen och koden.

För att uppnå detta valdes databasabstraktionsbiblioteket Doctrine. Doctrine är en så kallad ORM (Object Relation Mapper). ORM gör databasresurser till objekt och sköter länkningen mellan dessa. Som grund för en ORM står DBAL.

DBAL abstraherar kontakten till en databas till den grad att man inte behöver bry sig om vilken typ av databas som används på servern. Detta underlättar utvecklingen eftersom man kan ha en utvecklingsserver i Windows-miljö med t.ex. Microsoft SQL-server och en produktionsserver med MySQL.

ORM konfigureras med ett schema. Schemat består av en fil som innehåller instruktioner om hur databasen skall byggas upp. I Symfony kan databasen beskrivas genom att använda kommentarer i en entitetfil, i XML-filer eller i YAML-filer. För mitt projekt har jag valt att använda kommentarer i entitetfilen. Detta val gjordes eftersom syntaxen för Doctrine-kommentarer är väldigt lik den använd av dokumenteringsverktyget PHPDoc. Mer om detta finns i underkapitlet 2.1.3.

I Symfony kan man antingen definiera en sådan här entitetfil manuellt, generera den från en färdig databas eller använda inbyggda verktyg som fungerar via Symfonys kommandotolkgränssnitt. När man har entitetfilen färdig kan man generera funktioner för att komma åt dessa variabler. De här metoderna kallas för getters och setters. I MVC-synsättet är entitetfilen en Model. Ett exempel på en entitetfil finns i Bilaga 1.

Arbetsflödet jag valt är att först generera entitetfilerna via kommandotolkgränssnittet. När filerna är genererade kan jag sedan manipulera dem ytterligare. Varje gång man lägger till variabler i entitetetfilerna måste databasen uppdateras så att fält läggs till. Denna operation går att utföra automatiskt via kommandotolkgränssnittet.

Eftersom jag valt ramverket Symfony finns det egentligen bara två ORM:s att välja mellan, Doctrine och Propel. Skillnaden mellan Doctrine och Propel är väldigt liten men Doctrine är mer använt för Symfony (Potencier, 2009). Dessutom har jag använt Doctrine förut så jag är van med syntaxen och de bakomliggande funktionerna.

Eftersom databasens struktur är definierad i filer är det lätt att föra över hela projektet till en ny server. När man har fått filerna överflyttade kör man följande kommando i kommandotolken:

```
"php app/console doctrine:schema:update –force"
```

Kommandot kör en metod som läser in alla entitetfiler och bygger upp databasen i enlighet med dem.

2.5 REST-tjänst

Trots att klienterna i det här fallet inte är vanliga datorer kommer kontakten till servern i praktiken bestå av en webbtjänst. Eftersom klienten använder sig av en strömsnål mikrokontroller med relativt låg kapacitet får inte tjänsten kräva för mycket av processorn eller av minnet. Kommunikationen sköts också över GPRS vilket är relativt dyrt. Mer om detta finns i underkapitel 3.4.5.

REST står för Representative State Transfer. T.ex. om klienten skickar "färg = blå" till saker.com/bollar med verbet POST gör servern ett nytt objekt i databasen av typ boll och sätter strängen "blå" i objektets färgvariabel (Richardson&Ruby 2007 s.218). REST baserar sig på HTTP/1.0 och utvecklades i samband med HTTP/1.1. Eftersom kommunikationen med Sim900 via GPRS ändå bygger på HTTP, för REST med sig minimalt med extra data.

Det finns ett flertal sätt att göra webbtjänster på, varav SOAP och REST är de vanligaste (Wagh& Dr. Thool 2012). SOAP lämpar sig inte för det här ändamålet p.g.a. att:

- SOAP baserar sig på XML vilket är rätt processorintensivt att parsa.
- XML kräver också mycket överloppsdata vid överföring
- SOAP är ett komplicerat protokoll som innehåller många funktioner som inte går att använda på en mikrokontroller av denna storlek.

REST däremot har nästan ingen överloppsdata överhuvudtaget och går relativt enkelt att parsa med en mikrokontroller. I princip kunde jag utveckla egna protokoll eftersom jag bygger klienterna själv. Datamängden som skulle sparas är minimal och projektets komplexitet skulle öka p.g.a. detta vilket gjorde ett eget protokoll till ett olönsamt alternativ.

2.6 Versionshantering

Även om det finns enbart en kodare och utvecklare är det ändå nödvändigt att använda något system för att hålla reda på koden. Ett versionshanteringssystem hjälper en att hålla koll på sin kod och föra ett register över alla förändringar. Ett versionshanteringssystem hjälper också när flera personer arbetar med samma projekt och t.o.m. samma filer. Eftersom alla förändringar kan spåras rad för rad kan flera filer som modifierats samtidigt sammanfogas.

Det finns en handfull olika versionshanteringssystem som erbjuder lite olika funktioner. Dessa krav lades på versionshanteringssystemen för detta projekt:

- Måste fungera i den utvecklingsmiljö som används.
- Måste stöda förändringar per rad i koden och inte bara hela filer i gången.
- Måste vara gratis att använda.

Kodmassan i projektet består av två olika delar, koden för servern och koden för klienten. Koden skrivs i två olika utvecklingsmiljöer, Netbeans och MPLAB X. För koden på servern var valet av versionshanteringssystem enkelt. På servern används ramverket Symfony som i sin tur använder GIT. Det finns stöd för GIT i Netbeans men eftersom utvecklingen också görs med hjälp av kommandotolken över SSH var det lättare att utföra GIT-kommandon där.

Genom att ha serverkoden på en centraliserad server blir det också lättare att föra över kod från testservern till produktionsservern med minimala avbrott i tjänsten. Endast den kod som är skriven för projektet behöver överföras. Ramverket Symfony, alla tilläggsmoduler och alla kodsnuttar som använder sig av samma versionshanteringssystem behöver inte laddas ner från testservern. Eftersom dessa kodsnuttar förvaras på olika kodförvaringstjänster kan de laddas ner direkt därifrån.

För klienten använder jag mig av utvecklingsmiljön MPLAB X. Även om MPLAB X bygger på Netbeans fanns det inte stöd för GIT där. Däremot finns det stöd för Mercurial vilket också var lämpligt för projektet.

GIT och Mercurial är båda program med öppen källkod. För att kunna använda dem effektivt behöver man en centraliserad server som tar emot förändringar av utvecklarna. Det går att göra

själv men det finns också tjänster på nätet som är gratis. Jag har för det mesta använt mig av Github som är det mest använda versionshanteringstjänsten för kod (Finnley, 2011).

Github saknar stöd för Mercurial och begränsar användare till att endast ha ett privat projekt kostnadsfritt på sin tjänst. Därför måste en annan tjänst hittas för projektet. Bitbucket valdes eftersom det är gratis och stöder både Mercurial och GIT. Det är en klar fördel att ha all kod på samma ställe. Att hålla koden på en extern tjänst medför också ett extra lager av säkerhetskopiering eftersom koden alltid finns att hämta.

2.7 Automatisk testning av programvarukod

För att testa programkod kan man skriva och utföra automatiska test. Dessa test är små program som tar in data och kör den igenom metoder i mjukvaran som testas. Ett förbestämt förväntat resultat jämförs mot det som den testade metoden returnerar. Om det förväntade resultatet och det returnerade resultatet avviker misslyckas testet.

All kod måste testas. Det går att göra dessa test på förhand genom att t.ex. ladda en sida i en webbapplikation och se om det fungerar. Att manuellt testa all kod varje gång man ändrar någonting är tidskrävande och mänskliga fel uppstår lätt, varför man hellre förlitar sig på automatiska test. Att skriva automatiska test är tidskrävande men kan spara resurser på lång sikt. Automatisk testning kan användas som ett verktyg för verifiering och validering.

Med verifiering anses att man ser till att förutbestämda krav uppfylls eller så kallade funktionella test. Test kan skrivas för att se till att det finns en inloggningsruta som fungerar eller att det finns en meny med förutbestämda element. Dessa test kan skrivas före själva koden skrivs och därefter skrivs kod för att uppfylla testet.

När automatiska test används som validering av kod skriver man test som testar själva programlogiken. Dessa test kallas också för strukturella test. T.ex. kan man skriva ett test som ser till att en användare blir registrerad när rätt information skickas till servern.

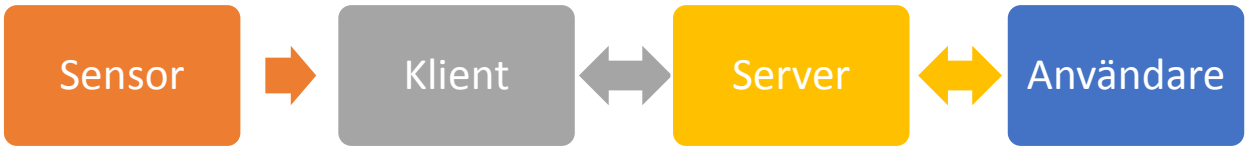
Automatisk testning används i det här projektet enbart för serverkoden. Klientens mjukvara har relativt lite kod och det är svårt att automatisera testningen. Därför testas den manuellt. Den del av serverns olika användargränssnittskod som körs i användarens webbläsare är skriven i JavaScript.

Eftersom dessa gränssnitt är lätta att testa manuellt och eftersom de ej påverkar varandra användes inte automatisk testning här.

Serverns kod bygger såsom tidigare nämnts på ramverket Symfony. Ramverket Symfony använder sig av testramverket PhpUnit för att testa koden. PhpUnit är ett av de vanligaste testramverken för PHP och är också en de facto-standarder för testning av PHP-kod. Därför var det ett logiskt val även för detta projekt.

3 PRODUKTUTVECKLING

Detta kapitel dokumenterar och motiverar de val som gjorts i produktutvecklingen. Kapitlet är uppdelat enligt de olika delarna som projektet består av. Kedjan av delar som projektet består av kan ses i Figur 1.



Figur 1. Projektets delar

En kort beskrivning av Figur 1 och därmed hela projektets dataflöde: Klienten får data från sensorn som klienten sedan skickar till servern. Användaren får data från servern och kan också mata in inställningar via användargränssnittet på servern. Servern kan svara på data som klienten skickar med nya inställningar. De nya inställningarna valideras och sparas i klienten.

3.1 Användargränssnitt

Det finns två användargränssnitt på servern: Kundgränssnittet och administratorgränssnittet. Det här kapitlet handlar om vilka krav som sätts på dessa gränssnitt och hur dessa krav har uppfyllts. Kapitlet fungerar som en gemensam grund för dessa användargränssnitt medan kapitel 3.2.1och 3.2.2 går in mer i detalj om hur dessa gränssnitt implementerats.

3.1.1 Översättning

Eftersom målgruppen för användargränssnittet pratar olika språk måste sidan också vara flerspråkig. Symfony har färdigt inbyggt stöd för flerspråkiga texter och denna funktionalitet implementeras redan från början. Vid utvecklingsskedet av sidan skrivs alla texter på ett språk men alla dessa textsträngar hamnar i en skild språkfil. När sidan är klar och de andra språken skall implementeras är det bara att kopiera och översätta denna fil.

Symfonys tilläggsmodul för översättning har stöd för språkfiler skrivna i PHP, YAML och XML. För det här projektet valdes YAML-formatet. YAML är ett filformat som bygger på text som är formaterad med speciella tecken, radbyten och indragningar. Ett exempel på en språkfil formaterad i YAML:

```
node:
    id: Id
    alias: Alias
    imei: Imei
    actions: Toiminnot
    show: Näytä
    add: Lisää anturi
table:
    timestamp: Päivämäärä
    battery: Akku
    signalstrength: 'Signaalin voimakkuus'
    signalquality: 'Signaalin laatu'
    sensor: Anturi
    event: Tapahtuma
event:
    update: Päivitys
    button: Nappi
    start: Käynnistys
menu:
    nodes: Anturit
    addnode: 'Lisää anturi'
    logout: 'Kirjaudu ulos'
    node:
        status: Tilanne
        graph: Kaavio
        table: Taulukko
        settings: Asetukset
```

YAML är ett filformat som är lätt att läsa och redigera. Därför var det ett naturligt val av filformat.

3.1.2 Navigering

För att göra användargränssnittet begripligt delas olika funktioner upp i olika sidor. För att navigera mellan dessa sidor används en meny. I menyn finns även användarens alla klienter för att öka överskådligheten och göra navigeringen snabbare. I de flesta fall har användaren bara en klient. Då är det ett onödigt steg att gå via en sida som skapar en lista på klienterna förrän man kommer till den specifika klientens sida.

För att generera menyerna används KnpMenuBundle som är en tilläggsmodul skriven för Symfony av KNPLabs. Genom att använda denna tilläggsmodul är det lätt att generera HTML-menyer som har rätt format för Bootstrap. Med hjälp av KnpMenuBundle kan menyer specialanpassas för specifika användare. T.ex. såsom tidigare nämnts kan en lista på användarens klienter visas i menyn.

Symfonys översättningsmodul är också implementerad i KnpMenuBundle vilket gör det lätt att översätta menyer. Symfony gör det också lätt att köra över standardstilmallar med egna stilmallar. På så sätt kunde stilmallen för kundgränssnittet implementeras också för menyerna. Mer om detta finns i underkapitel 3.2.1.

3.1.3 Optimering av klientkod

För att minimera laddningstid och serverbelastning hålls klientkodens filstorlek liten. För att uppnå detta används ett flertal knep. Dessa knep stöder sig på planering, på funktionalitet i Symfony samt på inbyggda hjälpmedel i moderna webbläsare.

För det första laddas inte onödiga kodbibliotek. De kodbibliotek som alltid krävs laddas på alla sidor medan andra laddas efter behov. Sidmallsmärkspråket Twig är till stor nytta för detta ändamål.

Varje sida har en egen Twig-stilmall. Dessa stilmallar kan i sin tur bygga på en annan stilmall och den i sin tur på en tredje o.s.v. På det här sättet bildar stilmallarna en trädstruktur där varje stilmall ärver parametrar från den tidigare stilmallen i trädstrukturen. Nyttan med denna trädstruktur är att man inte behöver skriva kod flera gånger. T.ex. kan det finnas en stilmall i roten av trädet som innehåller alla de viktigaste HTML taggarna och en grundläggande struktur. Nästa fil i

trädstrukturen kan innehålla element som används i en viss sektion. Den sista filen i länken är specifik för en viss sida och innehåller bara de element som är unika för just den sidan.

Förutom att man inte behöver skriva samma kod flera gånger hjälper Twig också med att ladda rätt JavaScript- och CSS-filer på rätt ställe. Filer såsom Bootstrap och Font Awesome behövs på så gott som varje sida. Därför laddas de redan i roten av trädstrukturen. Tilläggsbibliotek såsom datavisualiseringsbiblioteket Flot krävs bara på enstaka sidor och kan därför laddas endast där de behövs.

Om ett kodbibliotek består av flera filer kan dessa filer automatiskt slås ihop med hjälp av Twig och tilläggsmodulen AsseticBundle som kommer med Symfony. Nyttan med att slå ihop filer är att antalet förfrågningar till servern minskar. Dessutom kan man i samband med i hopslagningen köra filerna igenom andra processer.

En av dessa processer är YUI Compressor som är gjord av företaget Yahoo. YUI komprimerar CSS- och JavaScript-filer genom att förkorta variabelnamn och genom att ta bort allt som inte är vitalt för att de skall kunna läsas eller utföras. Detta innefattar kommentarer, radbrytningar, onödiga mellanslag m.m.

För att servern inte skall behöva lägga ihop alla sidmallfiler, utföra alla optimeringsprocesser o.s.v. varje gång en sida visas, mellanlagras alla sidor på serverns hårdskiva. När användaren skickar en begäran till servern för en viss sida kontrollerar servern först om det finns en fräsch kopia av den begärda sidan på hårdskivan. Om en tillräckligt ny mellanlagrad kopia hittas skickas den till kunden. I annat fall generas den mellanlagrade kopian på nytt. Mellanlagringssystemet i Symfony är automatiskt och lättanvänt. Sidor behöver inte genereras på nytt helt och hållet på grund av Twigs trädstruktur. Bara de delar där data byts ut behöver genereras på nytt.

Stora bibliotek såsom Bootstrap och Font Awesome laddas från en CDN så att belastningen på projektets server hålls liten. För Bootstrap och Font Awesome används BootstrapCDN som är gratis. Genom att ladda dessa kodbibliotek från en tredje part med stora serverresurser kan man vara säker på att de filer man behöver alltid är tillgängliga och att bandbredden räcker till även vid hög trafik.

Filer som inte ändras mellanlagras i webbläsaren. Det betyder att filerna inte laddas på nytt varje gång man laddar om sidan eller navigerar till en ny sida som använder samma filer. Detta sparar serverresurser och nätverksresurser. Skillnaden mellan att ladda en sida med och utan mellanlagring kan ses i Tabell 1.

Tabell 1. Jämförelse mellan sidladdning med och utan mellanlagring (detaljerad information i Bilaga 4 och 5)

Mellanlagring aktiverad	**Datamängd överförd totalt**	**Tid för laddning av sidstruktur**	**Total laddningstid**
Nej	580KB	580ms	1.03S
Ja	54KB	342ms	614ms

3.2 Server

Det här underkapitlet handlar om hur programvaran på servern byggts upp och vilka val som gjorts för de olika gränssnitt servern består av.

Serverns roll i projektet är:

- Att skicka meddelanden till användaren i larmsituationer via SMS, e-mail eller andra kommunikationsmetoder.
- Att erbjuda kunden ett kundgränssnitt för att administrera klienten, ställa in vart larmet skall skickas och visa data som kommer från klienten. Mer om detta finns i avsnitt 3.2.1.
- Att erbjuda ett klientgränssnitt för att ta emot data från klienten och skicka tillbaka inställningar. Mer om detta finns i avsnitt 3.2.3.
- Att erbjuda ett administratorgränssnitt där uppdragsgivaren kan ha översikt över alla klienter och användare. Mer om detta finns i avsnitt 3.2.2.

3.2.1 Kundgränssnitt

Kundgränssnittet följer de allmänna krav som definierats för användargränssnitt i underkapitlet Användargränssnitt. För kundgränssnittet har ytterligare funktionella krav ställts. Dessa krav har kommit fram genom samtal med uppdragsgivaren. Via gränssnitten skall kunden kunna:

- Se den data som skickas från klienten visualiserad.
- Ändra inställningar för klienten såsom mätintervall och geografisk position.
- Ändra inställningar för sitt eget användarkonto såsom namn, e-mail osv.
- Ändra vart larmet skickas och via vilka medier.

Meddelandet som kommer in från klienten består av olika värden och händelser samt en tidsstämpel för när meddelandet kom in. På grund av denna kombination av data och tidsstämpel är det lämpligt att visualisera klientens data i en graf och i en tabell. Genom att visualisera data med ett linjediagram är det lätt att få en överblick över hur olika värden har fluktuerat med tiden och hur de förhåller sig till varandra. För att få en mer detaljerad vy över en specifik händelse används en tabell. I en tabell är det lätt att se ett exakt värde vid en specifik tidpunkt. Dessa två visualiseringsmetoder kompletterar varandra väl.

För att göra någonting så komplext som att visa linjediagram på webben behövs en hel del kod. Det finns ett flertal kodbibliotek som kan användas för ändamålet. Därför är det inte lönsamt att utveckla själv. Kriterierna för kodbiblioteket som valdes är:

- Kodbiblioteket måste vara gratis att använda i kommersiellt syfte.
- Det måste finnas stöd i kodbiblioteket för ett flertal olika diagram såsom pajdiagram, stapeldiagram och linjediagram för att kunna ge ett enhetligt intryck överallt på användargränssnittet ifall att ett annat visualiseringssätt krävs.
- Det måste gå att anpassa det visuella såsom färger, skuggor, fonter och former så att diagrammet passar in i resten av det visuella temat.
- Koden måste kunna köras i alla moderna webbläsare utan att användaren behöver ladda ner tilläggsmoduler till webbläsaren.

För att hitta ett lämpligt kodbibliotek för visualisering av data som uppfyller dessa krav användes webbsajten Socialcompare (Socialcompare, 2013). Sajten har en utförlig jämförelse över olika

kodbibliotek för ändamålet. Efter en första genomgång låg valet mellan Google Chart Tools och FlotCharts varav det senare alternativet valdes.

Googles kodbibliotek är vida använt och väl testat och det var bekant från tidigare. Efter att ha testat det med riktig data konstaterades att det var problematiskt att få t.ex. linjediagram att passa in i det övriga utseendet. Dessutom lämpar sig inte Googles kodbibliotek för en responsiv design eftersom dess diagram inte skalar sig enligt storleken på användarens skärm.

FlotCharts är ett kodbibliotek som bygger på öppen källkod och är skrivet i JavaScript och jQuery. Kodbiblioteket uppfyller alla krav ovan och är lätt att använda. Det finns även tilläggsmoduler till Symfony som gör det möjligt att fylla diagrammet med dynamisk data. Ett exempel på ett diagram gjort med Flot kan ses i Figur 2.

Figur 2. Exempel på ett diagram gjort med kodbiblioteket Flot

Att visa data i en tabell är en av de mest triviala saker man kan göra på en webbapplikation som bygger på HTML. För uppgiften valdes ändå ett kodbibliotek eftersom en del tilläggsfunktionalitet

krävdes utöver att enbart visa data. Klienten skickar i medeltal in 10 meddelanden per vecka. Därför blir datamängden ganska snabbt relativt stor. Att visa hundratals rader i en tabell är inte användarvänligt och tar länge att ladda. För att göra tabellen mer användarvänlig och minimera laddningstiden kan man göra följande:

- Dela upp dataraderna i delar och visa en del i gången.
- Filtrera data så att t.ex. endast larmsituationer visas.
- Ladda data dynamiskt från servern utan att ladda om sidan.
- Låta användaren välja i vilken ordning data visas. T.ex. enligt tid eller sensornivå.

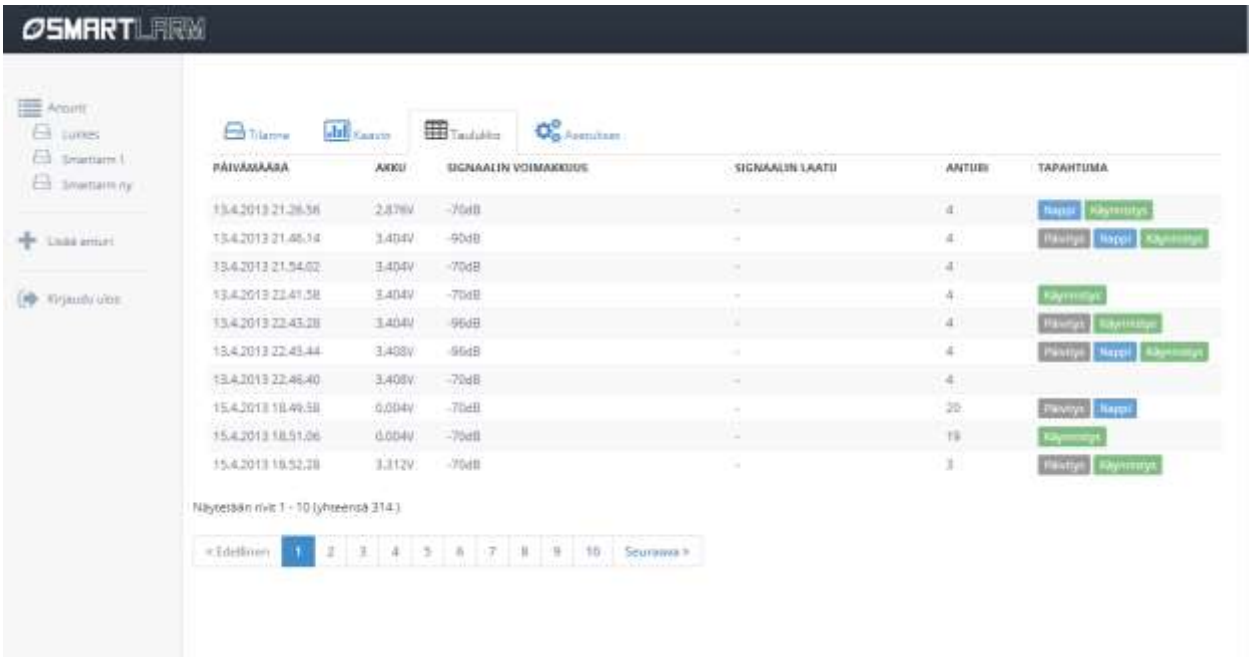
Kodbiblioteket som valdes för detta ändamål är Datatables som bygger på jQuery. Det finns inte så många alternativ när det kommer till att välja kodbibliotek av den här typen. I och med att jQuery används på andra håll i kundgränssnittet behövde också det här kodbiblioteket bygga på jQuery för att undvika att flera JavaScript-ramverk behövs.

Datatables har stöd för Bootstrap-stilmallar vilket gör det enkelt att integrera tabellen på sidan. Dessutom är det lätt att använda dynamisk data med Datatables och Symfony. AliDatatableBundle är en tilläggsmodul för Symfony som gör ett dedikerat datagränssnitt för Datatables. Gränssnittet har stöd för sökning, uppdelning av data, omformatering av t.ex. datumfält m.m.

För att utveckla de grafiska element som behövs för kundgränssnittet användes färdiga sidmallar. En av de största orsakerna till detta är att spara tid och pengar genom att inte behöva lägga ner tid på att utveckla någonting som andra med bättre kunskap i området redan har gjort.

För att välja sidmall användes nättjänsten Bootswatch. Tjänsten specialiserar sig på sidmallar för Bootstrap. På deras sida kan man bläddra igenom och testa sidmallar som är ämnade för användargränssnitt där användaren har en administrerande roll. Tillsammans med uppdragsgivaren kunde en sidmall med lämpligt utseende och en passande funktionalitet väljas.

Sidmallen som valdes heter DetailAdmin och är gjord av Eric Alvarez (Alvarez, 2013). Sidmallen har en färdig struktur med en balk högst uppe och en meny till vänster. Stilar för tabeller och diagram gjorda med kodbiblioteket FlotCharts fanns också färdigt. Därför var det lätt att presentera data på det sätt som uppdragsgivaren ville ha det. Ett exempel på DetailAdmin-stilmallen samt Datatables kan ses i Figur 3.



Figur 3. Exemepel på sidmall och tabell gjord med Datatables

3.2.2 Administratorgränssnitt

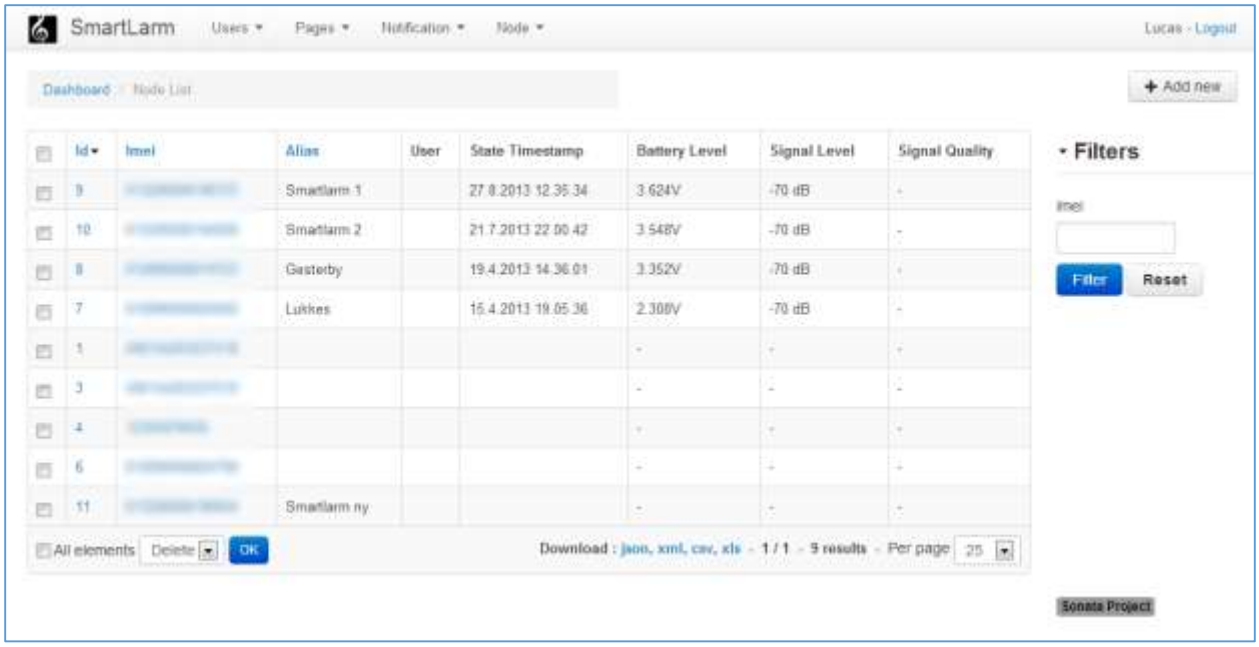
Administratorgränssnittet är ett webbgränssnitt ämnat att användas enbart av uppdragsgivarens personal. Via gränssnittet skall man kunna komma åt all data som finns i projektets databas och ändra på den vid behov. Genom att komma åt projektets databas på ett enkelt sätt kan man åtgärda fel och få en överblick över systemet.

Hela administratorgränssnittet är uppbyggt med tilläggsmodulen AdminBundle av Sonata. Sonata är en grupp mjukvaruutvecklare som fokuserar på att utveckla tillägg för Symfony. AdminBundle är ett verktyg för att visa data från databasen på ett innovativt sätt med sökfunktioner och tabeller.

Från administratorgränssnittet kan man också exportera data i CSV-format så att man kan skriva ut den eller öppna den i andra program. Den här funktionen är praktisk i kombination med sökfunktionen. T.ex. om man vill skriva ut en lista på alla klienter med lågt batteri gör man först en sökning på det och sedan skriver man ut en lista på dem.

Det går också att utveckla AdminBundle. När man t.ex. visar en lista på klienter kan man sätta in en symbol som visar signalstyrkan istället för att skriva ut den i dB. När man trycker på en klient kan man skräddarsy vilken information som visas och vilken information som kan ändras. Sonata

AdminBundles användargränssnitt bygger på ramverket Bootstrap på samma sätt som användargränssnittet för kunderna. Ett exempel på en vy från sonataAdminBundle kan ses i Figur 4.



Figur 4. Exempel på SonataAdminBundle

3.2.3 Klientgränssnitt

Klienterna kopplar upp sig till server via ett datagränssnitt. Kommunikationen från klienten till servern sker över GPRS och TCP/IP-protokollet som en HTTP-förfrågan. Klienten har låg processorkraft och ett litet minne. Dessutom är GPRS en relativt dyr kommunikationsmetod. Därför måste datamängden hållas liten. För att minimera mängden överförd information valdes headern x-www-form-urlencoded. Headern bestämmer hur datan skall formateras och hur hela meddelandet skall se ut.

Det är möjligt att skicka vilken data som helst över TCP/IP och paketet kunde förminskas ytterligare. Problemet med en sådan lösning är att TCP/IP-paketet inte skulle hitta fram till gränssnittet utan ytterligare kod. X-www-form-urlencoded är samma format som används när man skickar information från ett generiskt HTML-formulär till en server. Genom att använda samma format behöver inte ytterligare logik skrivas. Ramverket Symfony har också stöd för att verifiera

att de meddelanden som kommer in till servern är rätt strukturerade och har rätt data i sig. På så sätt förhindras felinmatad data. Mer om detta ämne kan hittas i underkapitel 3.4.

3.2.4 Användarrättigheter

Alla användare kommer åt systemet genom att registrera sig och använda sitt användarnamn och lösenord. Denna funktionalitet finns färdigt inbyggd i ramverket. För att hålla reda på vilka användare skall komma åt vilka klienter används ACL. ACL (Access Control List) eller tillgångskontrollistor är ett sätt att ge rättigheter till olika användare och grupper för tillgång till olika resurser.

Ursprungligen användes termen ACL för att beskriva hur operativsystem kan hålla reda på vilka rättigheter systemanvändare har att öppna och ändra på filer i filsystemet. Inom Symfony är ACL en generell term för att knyta användare och användargrupper till objekt. I det här fallet avses användare som finns i databasen och deras rättighet att manipulera och se andra resurser i databasen.

ACL är i grund och botten bara några databastabeller med tillhörande logik. Ofta skriver man denna logik själv men med Symfony är det onödigt. Om man bara vill knyta en användare till ett objekt är ACL onödigt komplicerat men i det här projektet krävs det avancerade rättighetssystem.

T.ex. om ett företag har hundratals klienter i olika kommuner kanske man vill ge företagets personal rättighet att administrera över alla dessa klienter. Utöver detta behöver montörer och annan personal som har hand om klienter inom ett visst område åtkomst till dem. Slutligen behöver slutkunden kunna administrera sin egen klient. Alla dessa parter kan behöva egna rättigheter för olika funktioner och olika gränssnitt. När saker blir mer komplexa är det bäst att utgå från färdiga lösningar som man vet att fungerar.

3.2.5 Kontakt med kunden

Vid larmsituationer skall kunden kontaktas via epost eller SMS. Meddelandet formateras med hjälp av olika Twig-mallar. Eftersom Symfonys sidmallssystem kan användas till annat än bara HTML går det bra att formatera e-post och SMS-meddelanden.

Epost skickas ut med hjälp av biblioteket Swiftmail som finns att hämta som en tilläggsmodul till Symfony. I meddelandet finns en länk som kunden kan besöka för att kvittera larmet och för att se klientens status.

För att skicka ut SMS används en SMS-nätslusstjänst *(eng: SMS gateway service)*. Servern skickar meddelandet till tjänsten och därifrån skickas SMS:et ut till kunden. Genom att svara på SMS:et kan kunden kvittera larmet. När kunden svarar på meddelandet går svaret till SMS-tjänsten. SMS-tjänsten i sin tur kontaktar servern via ett dedikerat gränssnitt. SMS:et kan också innehålla en länk precis som e-postmeddelandet för kunder med smarttelefoner.

Det finns en handfull SMS-tjänster. För att välja en tjänst som passar projektets behov fastslogs följande krav:

- Tjänsten måste vara kostnadseffektiv.
- Tjänstens måste ha funnits en längre tid på marknaden för att minimera chansen att tjänsten upphör.
- Tjänsten måste erbjuda ett gränssnitt för kommunikation mellan servern och tjänsten.
- Detta gränssnitt måste vara väldokumenterat och lätt att utveckla kod för.

Tjänster som uppfyllde dessa krav var Nexmo och Twilio. Information om dessa tjänster hittas på www.nexmo.com (Nexmo, 2013) respektive www.twilio.com (Twilio, 2013). Skillnaden mellan dessa tjänster är marginell. Twilio erbjuder förutom SMS-tjänster olika tjänster uppbyggda kring röstkommunikation. Nexmo erbjuder också en del röstbaserade tjänster men de är inte lika långt utvecklade.

Eftersom det här projektet inte behöver sådana tjänster skapar detta bara onödig komplexitet. Nexmo är aningen billigare och dokumenteringen är lättare att förstå. Dessa faktorer ledde till att Nexmo valdes som tjänsteleverantör.

3.3 Klient

Klienten i det här sammanhanget är den hårdvara som utvecklas för projektet. Hårdvarans huvudkomponenter består av en mikrokontroller, en Sim900 GSM/GPRS-modul och en mätprob.

Själva elektroniken ingår inte enligt avgränsningen för det här examensarbetet. Det här kapitlet beskriver dessa komponenter och den data som strömmar mellan dem.

3.3.1 Sim900

Sim900 är en elektronisk hårdvarumodul för GSM-kommunikation gjord av Simcom. Modulen innehåller komponenter och mjukvara som möjliggör att data kan skickas som SMS eller GPRS. I det här skedet kommunicerar klienten enbart via GPRS. Data och kommandon skickas över en seriedataanslutning (Simcom, 2010). Sim900-modulen kan ses i Figur 5.

Figur 5. En Sim900-modul. (hämtad med tillstånd från: http://kamami.com/published/publicdata/BTC10/attachments/SC/products_pictures/Sim900.jpg)

Vi valde att använda en färdig modul för att spara på tid och utvecklingskostnader. Att själv bygga upp hårdvara för GPRS-kommunikation är väldigt krävande och man måste certifiera sin hårdvara.

Sim900-modulen har en hel del onödig funktionalitet för det här projektet såsom audio ut- och ingång samt stöd för knappsatser. På grund av att modulen ändå är så relativt billig medför detta inte några problem.

3.3.2 Mikrokontroller

Ett krav från uppdragsgivaren var att använda Microchips mikrokontrollers eftersom de är vana med deras ekosystem av kretsar, dokumentation och utvecklingsverktyg. Det här examensarbetet kommer inte att gå in på djupet om mikrokontrollers men vissa aspekter måste ändå beröras.

En mikrokontroller är en integrerad krets som innehåller en processor, arbetsminne, programminne samt diverse stödfunktioner. Stödfunktionerna kan vara t.ex ADC för att mäta spänningar eller timers för att räkna pulser eller mäta tid.

För det här projektet valdes Microchips PIC 16lf1829. Kretsen är optimerad för energisnåla applikationer med avancerade vilolägen och ställbara klockfrekvenser. Kretsen har också en

speciell timer som kallas för Watch Dog Timer (WDT). Timern kan kopplas till en inbyggd långsam klockkristall som går på 32 768 hz. För varje klockpuls adderas ett till en 16-bit variabel. När variabeln är fylld med 1-bitar väcks kretsen från viloläget.

För att utöka tiden som kretsen kan vara i viloläget ytterligare kan man ställa in hur många pulser från klockan skall registreras innan värdet ökas med ett. Maxinställningen för hur många pulser som måste räknas innan timern ökar med ett är 2^{23}. Detta värde kallas för prescaler-värdet. För att räkna ut hur länge kretsen kan hållas i viloläge används formeln prescaler/klockfrekvens. 2^{23} delat med 32768hz blir 256 sekunder. Alltså är den längsta tiden man kan hålla kretsen i viloläget 256 sekunder.

När kretsen försätts i viloläge händer följande saker:

- Alla oanvända stödfunktioner stängs av.
- Alla portar sätts antingen höga, låga eller som ingångar beroende på vad som är kopplat till dem för att inga läckströmmar skall uppstå.
- Watch Dog Timern (WDT) sätts igång.
- Klockfrekvensen körs ner från 4Mhz till 32khz

Största delen av tiden är mikrokontrollern alltså i viloläge. Var femte minut (ca 256 sekunder) vaknar den upp. När den vaknar upp kan den göra tre olika saker:

1. Gå direkt tillbaks i viloläge ifall man vill att den skall göra en mätning mer sällan än var 256:e sekund. T.ex. genom att göra denna manöver 5 gånger kan man göra en mätning varje halvtimme.
2. Göra en mätning och vid larmfall skicka in data. I annat fall gå tillbaks till viloläget.
3. Göra en mätning och skicka in data oberoende om de uppmätta värdena är över larmnivån eller inte. Den här funktionen används för att klienten skall skicka in uppdateringar till servern regelbundet men med ganska långa intervall, t.ex. en gång per dygn eller en gång i veckan. Genom att servern får regelbundna uppdateringar är det möjligt att kontrollera att problem inte uppstått med klienten.

En illustration av logiken visas i flödesdiagrammet i Bilaga 3.

3.4 Data och kommunikation

Det här underkapitlet beskriver kommunikationen som sker mellan klientgränssnittet på servern och klienten. Kort sagt tar klienten initiativ för kommunikation då den vaknar upp från viloläge och kontaktar då servern. Servern svarar klienten med ett HTTP-meddelande med statuskod 200. Eventuellt kan svaret från servern också innehålla nya inställningar för klienten.

3.4.1 Klientidentifiering

För att kunna identifiera vilken klient som kontaktar servern måste varje klient ha ett eget identifikationsnummer. Eftersom det är problematiskt att mata in unik data i varje klient används Sim900-modulens IMEI-nummer istället. IMEI-numret är en unik kod som identifierar Sim900-modulen. Med hjälp av IMEI-numret kan man skilja klienterna från varandra. IMEI-numret är 14 tecken långt plus ett kontrolltecken.

För extra kontroll läses också ICCID-numret från SIM-kortet. ICCID-numret är en unik kod för SIM-kortet. ICCID-numret är 19 tecken långt plus ett kontrolltecken. Genom att också skicka det till servern kan man kontrollera om SIM-kortet har bytts. Dessutom är det lättare att skydda sig mot förfalskade förfrågningar eftersom det nu krävs en 14+19 siffror lång kod som måste stämma överens med det som finns på servern.

Vid produktion av hårdvaran registreras alla IMEI-nummer och ICCID-nummer genom att läsa de streckkoder som finns på Sim900 modulen och SIM-kortet. Mikrokontrollern läser IMEI- och ICCID-numren via seriedataanslutningen.

3.4.2 Insamling av data

Data som skickas till servern tas från många olika källor. Det centrala i hela kommunikationen är sensorvärdet. Sensorvärdet kan hämtas in på flera olika sätt beroende på hur hårdvaran utformas. I fallet av en nivåvakt består sensorvärdet endast av en etta eller en nolla som representerar att nivån är under eller över en förutbestämd höjd. Mera avancerade sensorer kan också kopplas in så att ett analogt mätvärde kan avläsas.

Förutom mätdata och data som är kopplad till klientidentifiering krävs också att batteri- och signalstyrkan mäts. Batteristyrkan mäts med hjälp av mikrokontrollerns inbyggda ADC. Signalstyrkan kan läsas från Sim900-modulen via seriedataanslutningen. De här värdena samlas in för att kunna varna kunden ifall batterispänningen eller signalstyrkan blir för låg.

3.4.3 Inställningar på klienten

Klienten har vissa inställningar som man måste kunna modifiera via webgränssnittet på servern Genom att kunna modifiera vissa inställningar via servern kan man förändra klientens beteende utan att ändra dess programvara. Dessa inställningar är:

- Vilka sensorer som är aktiverade.
- Sensorernas larmtröskel.
- En flagga som berättar om mätvärdet skall vara större eller mindre än dess tröskel för att skicka ett larm
- Batterinivåns larmtröskel. Eftersom olika batterityper har varierande prestanda på låg batterinivå är det bra att kunna ställa in detta.
- Ett värde för hur många gånger klienten skall vakna upp förrän den gör något. Används för att förlänga vilolägestiden.
- Ett värde för hur många gånger klienten skall vakna upp förrän den skickar ett uppdateringsmeddelande till servern oberoende av om det uppstått en larmsituation eller inte.

Dessa värden har en standardinställning som är skriven i källkoden. Första gången programmet körs skrivs dessa värden in i mikrokontrollerns EEPROM. Om kunden eller en administrator vill ändra på inställningarna matas inställningarna in på servern.

För inställningarna som finns på servern respektive inställningarna som finns på klienten genereras varsin CRC-kontrollkod. Klienten skickar med sin CRC-kontrollkod till servern vid varje kommunikationstillfälle. Om klientens CRC-kontrollkod inte stämmer överens med serverns betyder det att det finns nya inställningar på servern. Då skickar servern de nya inställningarna till klienten i svaret på klientens förfrågan. I annat fall skickas ett tomt svar tillbaks för att klienten skall veta att meddelandet gick fram.

3.4.4 Förfrågan

Servern kan inte skicka förfrågningar till klienten eftersom den för det mesta är i ett viloläge för att spara ström. I viloläge är klienten inte uppkopplad på nätet och servern kan därför inte kontakta den. Den enda gången som data överförs mellan servern och klienten är när klienten vaknar upp och skickar en förfrågan till servern och när servern svarar på förfrågan.

För att spara på ström och överföringskostnader skickas bara en förfrågan. Förfrågans uppgift är:

- Identifikation med hjälp av IMEI- och ICCID-numret.
- Skicka över signalstyrkan, batterinivån samt sensorvärdet.
- Definiera vilken typ av händelse det är frågan om (batterilarm, sensorlarm, knapptryckning eller uppdatering).
- Skicka över CRC för inställningarna.

Förfrågan skickas som ett HTTP POST-meddelande. Ett exempel på meddelandet kan se ut såhär:

```
POST /node/api HTTP/1.1
Host: www.smartlarm.fi
Content-Type: application/x-www-form-urlencoded
Content-Length: 55


&IMEI=013226008196954&CSQ=10,0&BAT=3396&CCID=120311401851&EV=20&CRC=79&
S[0][v]=14649&S[0][e]=1&S[0][a]=0
```

Meddelandet består av ca 225 byte. De flesta operatörer debiterar per kilobyte. Ytterligare förminskningar av meddelandet skulle endast leda till marginella fördelar.

Eftersom hårdvaran stöder fler än en sensor är sensordata formaterad som en multidimensionell vektor. Första dimensionen har bara en post med nyckeln noll. Nyckeln är ett förutbestämt identifikationsnummer för en viss sorts sensor. Med hjälp av detta nummer vet servern vilken sensor det är frågan om.

Sensor noll har tre värden, v, e och a. V står för value eller värde och visar det oförändrade värdet från sensorn. Detta värde beror på vilken sensortyp det är frågan om. E står för error och har värdet ett vid felläge. Ett felläge kan uppstå om sensorn inte ger ut data inom utsatta gränser eller om kontakten till sensorn är bruten. A står för alarm och har värdet ett när mikroprocessorn tolkar v som ett alarmläge.

3.4.5 Abonnemang

För att kunna skicka data över GPRS-nätverket behövs ett abonnemang med tillhörande SIM-kort. För att kunna hantera en större mängd SIM-kort är det inte praktiskt att göra ett nytt avtal för varje enskilt SIM-kort. Därför anlitas en M2M-tjänsteleverantör som är specialiserad på att förse andra företag med SIM-kort och tilläggstjänster. M2M står för Machine to Machine eller maskin till maskin. I det här fallet är det fråga om kommunikation från klienten till servern. För att hitta en lämplig M2M-tjänsteleverantör har följande krav bestämts i samarbete med uppdragsgivaren:

- Tjänsten måste vara kostnadseffektiv. En stor del av kundens utgifter kommer från abonnemanget.
- Tjänsten måste ha ett API som tillåter vår server att kontakta den. Behovet av denna kontakt är att procedurer som t.ex. aktivering och deaktivering av SIM-kort kan automatiseras. Det skall också gå att beställa flera SIM-kort utan att kontakta tjänsteleverantören manuellt.
- Tjänsteleverantören måste ha alternativ både för vanliga SIM-kort av plast och för så kallade embedded SIM-kort som består av en integrerad krets som löds fast i kretskortet.
- Tjänsteleverantören måste vara väletablerad på marknaden. Om tjänsteleverantören går i konkurs slutar också vår produkt att fungera.

I skrivande stund har offerter kommit in från en handfull tjänsteleverantörer. Inget konkret val har ännu gjorts. Tjänsteleverantörerna skiljer sig mycket från varandra i prissättning och målgrupp. Stora företag såsom t.ex. Sonera kräver avgifter bara för att man skall få bli deras kund. De erbjuder all den funktionalitet som krävs för projektet men är inte kostnadseffektiva för ett relativt litet projekt som detta med under 20 000 kunder. Den andra extremen är företag som är väldigt förmånliga att arbeta med men som inte har den funktionalitet eller trovärdighet som krävs.

3.5 Produktens livscykel

Förutom att själva hårdvaran och serverlogiken utvecklas måste man också tänka på produktens livscykel. Med detta menas hur produkten tas i bruk, hur kontakten till kunden sköts, fakturering, underhåll, SIM-kort m.m.

3.5.1 Produkthelheten som en tjänst

Den förra produkten såldes som en vara medan den nya produkten skall säljas som en tjänst. Motiveringen av detta beslut är att produkten är såpass länge i bruk att det är svårt att med säkerhet säga vad som kommer att hända. Då är det lättare att bara skicka ut ny hårdvara ifall någonting går sönder och på så sätt blir det lättare för kunden.

Faktumet att den nya produkten behöver både SIM-kort och server, vilka båda är löpande kostnader, påverkade också beslutet att sälja systemet som en tjänst.

3.5.2 Ibruktagning

Alla klienter som skickas ut till kunden har ett eget IMEI-nummer som registreras i databasen. Klienten behöver inget eget serienummer eftersom IMEI-numret är unikt. Varje SIM-kort som säljs med klienten registreras också via dess ICCID-nummer. Tillsammans med mjukvaruversion och datum och dessa identifikationsnummer finns all data som behövs för att kunna spåra klienten ifall någonting går fel.

Kunden får klienten med SIM-kortet färdigt installerat. Alternativt kan SIM-kortet vara en integrerad krets på kretskortet. Det enda som behöver göras är att sensorn skall installeras i tanken och batteriet måste läggas på plats. Därefter tar klienten kontakt med servern och registrerar sig som tagen i bruk.

Därefter kan kunden registrera sig via kundgränssnittet. För att kunden skall kunna registrera sin egen klient och koppla den till sitt användarkonto måste IMEI-koden skrivas in. Kunden måste därefter gå igenom en enkel guide för att skriva in de viktigaste inställningarna för klienten.

Om Smartlarmsystemet säljs till större företag som sköter tömning av avloppstankar kanske de har egen servicepersonal som installerar klienterna. I så fall måste gränssnitt skapas för servicepersonal och e-mail skickas ut till kunderna så att de kan administrera sin egen klient. Eventuellt skall användargränssnittet förses med företagets logo och färger.

3.5.3 Underhåll av klienter

Mycket kan gå fel i klienten och dessa fel måste kunna upptäckas och åtgärdas. Servern får regelbundna uppdateringar från klienten och om en sådan uppdatering uteblir vet man att någonting har gått snett och då måste kunden uppmärksammas om detta.

De delar som är mest utsatta är batteriet och sensorn. Spänningen i batteriet mäts regelbundet och när den går under en viss nivå vet man att det är dags att byta ut det. Mikrokontrollern kan också märka om kontakten till sensorn bryts.

Genom att automatisera felsökningen så långt som möjligt sparar man tid och resurser. Systemet blir också säkrare genom att göra automatiska test. Det värsta som kan hända är att nivån i den tank man mäter går över eller under sin alarmnivå utan att kunden uppmärksammas om saken.

Om en klient går sönder är det lättast att byta ut den helt och hållet. När klienten byts ut måste det nya IMEI-numret registreras och all gammal data kopplas ihop med den. Servern måste därför ha verktyg i samband med administratorgränssnittet för att göra sådana förändringar på ett enkelt sätt utan att man måste gå in i databasen direkt och ändra på fält.

3.5.4 Hantering av versioner

Ifall av att uppdragsgivaren kommer ut med en ny produkt i framtiden måste servern byggas på ett sådant sätt att den kan arbeta med både ny och gammal hårdvara. Om uppdragsgivaren säljer produkten till ett större företag kanske de vill ha en egen version av hårdvaran och/eller mjukvaran. Dessa skillnader måste servern kunna ta hand om.

När IMEI-numret registreras sparas också hårdvarans och mjukvarans versionsnummer i databasen. Varje gång klienten tar kontakt med servern skickas IMEI-numret med och på så sätt kan servern skilja olika versioner och ta deras olika behov i beaktande.

3.5.5 Framtida utveckling

Tyngdpunkten för projektets användningsområde har varit slutna avloppstankar. Trots det kan produkten användas för många andra ändamål där enkla analoga eller digitala värden skall mätas,

loggas och skickas till en server. Dessa möjligheter kan man forska och spekulera i för att på så sätt styra utvecklingen på ett sådant sätt att sådana förändringar är möjliga.

Det är viktigt att redan nu göra val med tanke på framtiden så att man inte låser sig till att enbart göra en sak med systemet. En del potentiella kunder har redan hittats men ifall något stort företag blir intresserat kunde specialanpassade gränssnitt till systemet behövas. T.ex. om larmdata behöver integreras direkt i andra företags system kan ett API öppnas upp för detta ändamål.

I detta skede (oktober 2013) är det ännu oklart om kunden skall faktureras av oss direkt eller genom någon utomstående faktureringstjänst. Med tanke på mängden fakturor som måste skrivas om produkten säljer bra är det troligt att en utomstående faktureringstjänst anlitas. Då behöver gränssnitt för fakturering utvecklas på servern.

Utvecklingen av projektet kommer att fortsätta även efter examensarbetets slut. Den grund som har lagts kommer att komma till nytta långt in i framtiden. Även om detta projekt är en prototyp kommer så gott som all kod att användas i produktionsversionen.

Nästa steg är att projektet går över till betatestning med ett flertal klienter och testkunder. Först då kommer alla komponenter i projektet till sin rätt och eventuella fel dyker upp. Det är också då som nya behov för de olika gränssnitten kommer upp.

4 SLUTSATSER

Målsättningen med uppdragsgivarens projekt är att utveckla ett komplett produktpaket som kan säljas som en tjänst till privatpersoner och företag. Målsättningen med detta examensarbete är att definiera, utveckla och dokumentera denna produkt.

Kravställningarna för projektets olika delar är en viktig del av hur projektet utvecklas. Många faktorer påverkade kraven och hur de kan uppfyllas. Genom att metodiskt gå igenom varje enskild delkomponent i projektet och skriva om dem dök många faktorer upp som jag inte hade tänkt på från början.

I samband med att jag studerade ramverket Symfony studerade jag också andra mjukvaruutvecklares kod. Det hjälpte mig att se vilka komponenter som oftast behövs och vad

man bör tänka på. I själva verket var valet av ramverket ett av de mest avgörande valen och en stor del av projektets framgång.

Alla de gränssnitt som krävdes för servern är implementerade och fungerande. Ännu finns en del att finslipa men de är såpass färdiga att de kan visas upp för potentiella kunder. Hårdvaran för klienten är färdigt planerad och produktionsredo efter ett flertal prototyper. Mjukvaran för klienten är också fungerande men kräver ytterligare utveckling och testning.

Eftersom produktutvecklingen har lett till en fungerande produkt och en nöjd uppdragsgivare kan man konstatera att detta examensarbete har påverkat produktutvecklingen på ett positivt sätt.

KÄLLOR

American National Standards Institute. 2013, About ANSI Overview, Tillgängligt: http://www.ansi.org/about_ansi/overview/overview.aspx?menuid=1, Hämtad: 5.11.2013

Erick Alvarez. 2013, DetailAdmin - ResponsiveTheme, WrapMarket LLC, tillgängligt: Https://wrapbootstrap.com/user/erick75, Hämtad: 25.9.2013

Finnley, Klint. 2011, Github Has SurpassedSourceforge and Google Code in Popularity, Tillgängligt: http://readwrite.com/2011/06/02/github-has-passed-sourceforge#awesm=~ogB4UlwZSeOHgM, Hämtad: 5.9.2013

Github. 2013, PopularStarredRepositories, Tillgängligt: Https://github.com/popular/starred, Hämtad: 25.4.2013

Microchip. 2011, PIC16(L)F1825/1829 Data Sheet, Tillgängligt: http://ww1.microchip.com/downloads/en/DeviceDoc/41440B.pdf, Hämtad 24.4.2013

Moxcey, Mike. 2007, Documentingcode as youwrite is worth the hassle, Tillgängligt: http://www.techrepublic.com/article/documenting-code-as-you-write-is-worth-the-hassle, Hämtad: 12.9.2013

Nexmo, 2013, Nexmo - Cloud SMS and Voice API that connects you directly to carriers. Tillgängligt: https://www.nexmo.com, Hämtad: 21.11.2013

Nielsen, Jakob. 1995, 10 UsabilityHeuristics for User Interface Design, Tillgängligt: http://www.nngroup.com/articles/ten-usability-heuristics, Hämtad: 14.8.2013

Packagist. 2013, The PHP package archivist. Tillgängligt: https://packagist.org/, Hämtad: 21.11.2013

Pettit, Nick. 2012, Treehouse, Beginner's Guide to Responsive Web Design, Tillgängligt: http://blog.teamtreehouse.com/beginners-guide-to-responsive-web-design, Hämtad: 14.8.2013

Phpandstuff. 2009, Top 10 ReasonsWhyYouShouldUse a PHP Framework, Tillgänglig: http://www.phpandstuff.com/articles/top-10-reasons-why-you-should-use-a-php-framework, Hämtad: 24.3.2013

Potencier, Fabien. 2009, Doctrine vs Propel, Tillgänglig: http://symfony.com/blog/doctrine-vs-propel, Hämtad: 24.3.2013

Potencier, Fabien. 2011, What is Symfony2, Tillgänglig: http://fabien.potencier.org/article/49/what-is-symfony2, Hämtad: 24.3.2013

Python Software Foundation. 2012, PythonVsPhp, Tillgänglig: http://wiki.python.org/moin/PythonVsPhp, Hämtad: 24.3.2013

Richardson, Leonard; Ruby, Sam. 2007, RESTful Web Services, 1 uppl., Californien: O'Reilly Media, Inc., 422s.

Ritchie Dennis. 1993, The Development of the C Language. Bell Labs/Lucent Technologies. Tillgängligt: http://cm.bell-labs.com/who/dmr/chist.html, Hämtad: 5.11.2013

Rouse, Margaret. 2006, Access control list (ACL), Tillgängligt: http://searchsoftwarequality.techtarget.com/definition/access-control-list, Hämtad: 24.3.2013

Simcom. 2010, AT Commands Set SIM900_ATC_V1.00, Tillgängligt: http://www.simcom.us/act_admin/supportfile/SIM900_ATC_V1.00.pdf, Hämtad: 24.4.2013

Socialcompare.com. 2013, JavaScript Graphs and Chartslibaries, Tillgängligt: http://socialcompare.com/en/comparison/JavaScript-graphs-and-charts-libraries, Hämtad: 25.4.2013
Teutch. Moshe. 2012, Introduction to PhpDoc, Tillgängligt: http://www.sitepoint.com/introduction-to-phpdoc, Hämtad: 12.9.2013
The PHP Group, 2013, Historyof PHP, Tillgänglig: http://php.net/manual/en/history.php. Hämtad 24.3.2013
The PHP Group. 2013, HistoryofPhp, Tillgängligt: http://www.php.net/manual/en/history.php.php, Hämtad: 25.4.2013
Twilio. 2013, Twilio Cloud Communications - APIs for Voice, VoIP and Text Messaging, Tillgängligt: http://www.twilio.com, Hämtad: 21.11.2013
Wagh, Kishor; Dr. Thool, Ravindra. 2012, A ComparativeStudyof SOAP Vs REST Web Services ProvisioningTechniques for Mobile Host, Tillgängligt: http://www.iiste.org/Journals/index.php/JIEA/article/download/2063/2042, Hämtad: 24.3.2013
Wikibooks. 2013, Choosing the right programminglanguage, Tillgängligt: http://en.wikibooks.org/wiki/Web_Development/Choosing_the_right_programming_language, Hämtad: 14.6.2013
Vogel Peter. 2013, WhyYouShouldn'tComment (or Document) Code, tillgängligt: http://visualstudiomagazine.com/articles/2013/06/01/roc-rocks.aspx hämtad: 30.9.2013
Zafar, Rehman. 2012, What is software testing? Whatare the different typesoftesting?, Tillgängligt: http://www.codeproject.com/Tips/351122/What-is-software-testing-What-are-the-different-ty, Hämtad: 15.9.2013

BILAGOR

Bilaga 1. Exempel på en Doctrine-entitet

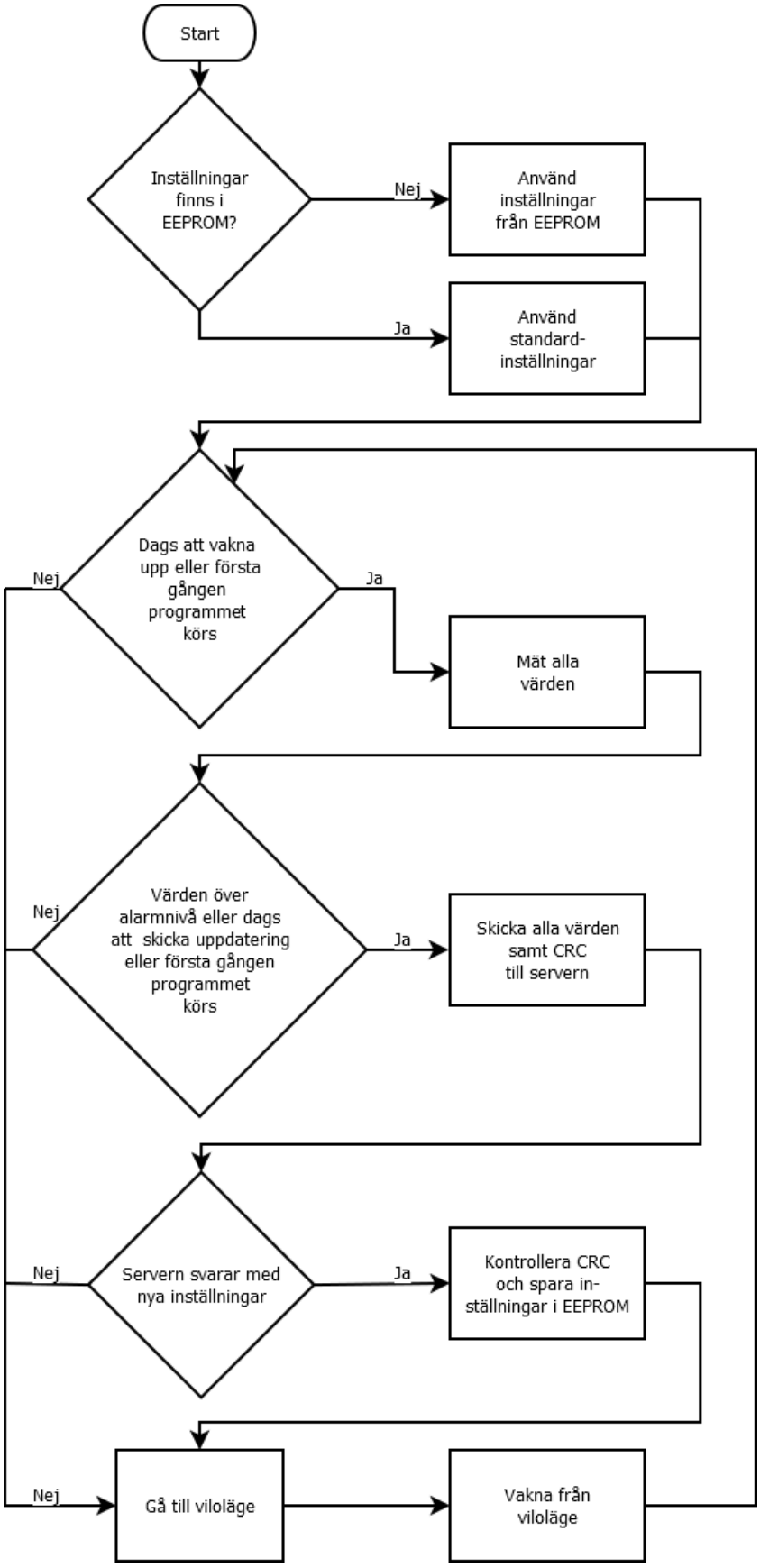
```
<?php
namespace Smartel\SmartLarmBundle\Entity;
use Doctrine\ORM\Mapping as ORM;
/**
 * Node
 * @ORM\Table(name="node__node")
 *
@ORM\Entity(repositoryClass="Smartel\SmartLarmBundle\Entity\NodeReposit
ory")
 */
class Node
{
    /**
     * @var integer
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;

    /**
     * @var string
     * @ORM\Column(name="imei", type="string", length=255)
     */
    private $imei;
    /**
     * @var string
     * @ORM\Column(name="alias", type="string", length=255,
nullable=true)
     */
    private $alias;
}
```

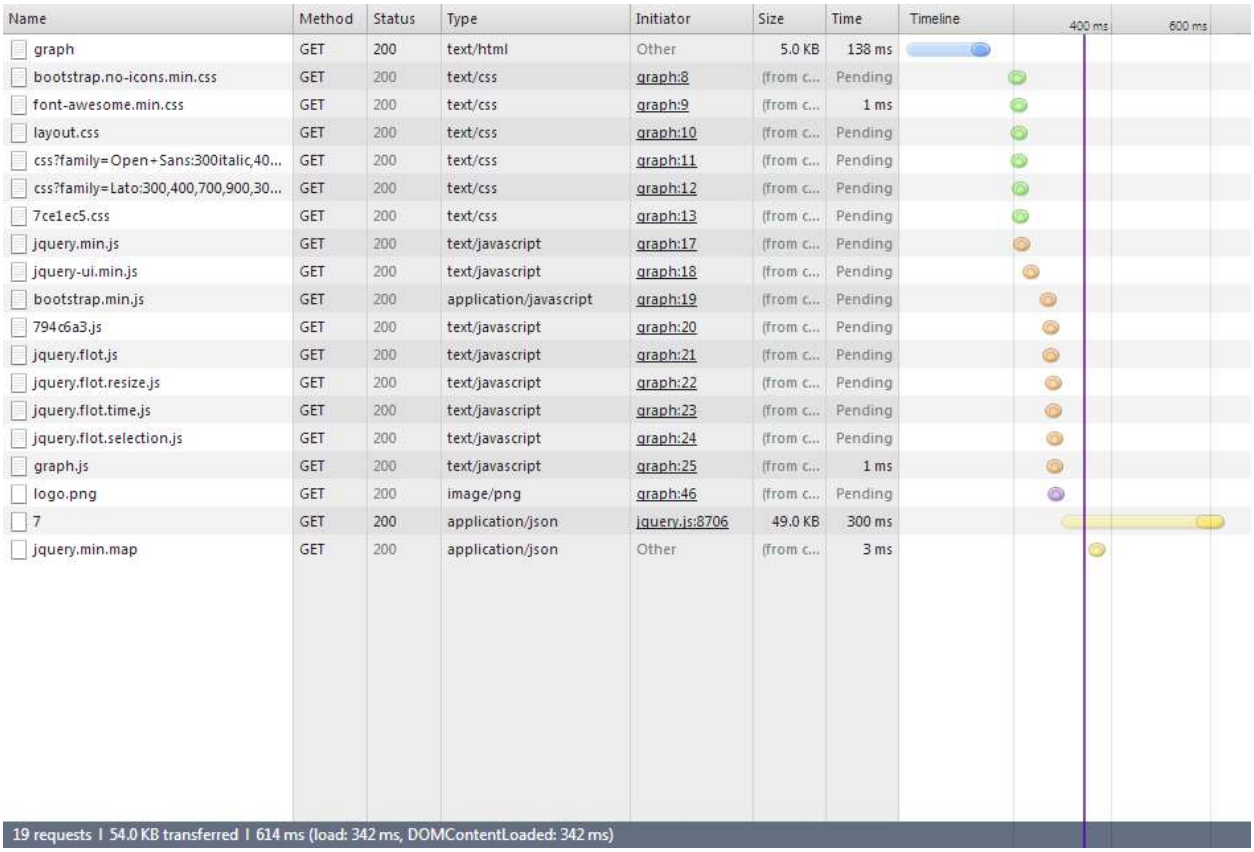
Bilaga 2. Lista på använda bundles

```
"php":                                          ">=5.3.3",
"symfony/symfony":                              "2.2.*",
"doctrine/orm":                                 "~2.2,>=2.2.3",
"doctrine/doctrine-bundle":                     "1.2.*",
"twig/extensions":                              "dev-master",
"symfony/assetic-bundle":                       "dev-master",
"symfony/swiftmailer-bundle":                   "2.2.*",
"symfony/monolog-bundle":                       "2.2.*",
"symfony/finder":                               "2.2.*@dev",
"sensio/distribution-bundle":                   "2.2.*",
"sensio/framework-extra-bundle":                "2.2.*",
"sensio/generator-bundle":                      "2.3.*",
"kriswallsmith/assetic":                        "dev-master",
"phpunit/phpunit":                              "3.7.*",
"symfony-cmf/routing-extra-bundle":             "1.0.*@dev",
"symfony-cmf/routing":                          "1.1.x-dev",
"sonata-project/user-bundle":                   "dev-master",
"sonata-project/page-bundle":                   "dev-master",
"sonata-project/seo-bundle":                    "1.1.*@dev",
"sonata-project/notification-bundle":           "2.2.*@dev",
"sonata-project/admin-bundle":                  "dev-master",
"sonata-project/doctrine-extensions":           "1.*@dev",
"sonata-project/doctrine-orm-admin-bundle":     "dev-master",
"sonata-project/block-bundle":                  "dev-master",
"sonata-project/cache-bundle":                  "dev-master",
"sonata-project/intl-bundle":                   "dev-master",
"sonata-project/easy-extends-bundle":           "dev-master",
"sonata-project/google-authenticator":          "1.*@dev",
"sonata-project/jQuery-bundle":                 "1.8.*@dev",
"sonata-project/exporter":                      "1.2.0",
"friendsofsymfony/user-bundle":                 "1.3.*",
"friendsofsymfony/rest-bundle":                 "dev-master",
"liip/theme-bundle":                            "dev-master",
"problematic/acl-manager-bundle":               "dev-master",
"knplabs/knp-paginator-bundle":                 "dev-master",
"knplabs/knp-menu-bundle":                      "1.1.*@dev",
"knplabs/knp-menu":                             "1.1.*@dev",
"craue/formflow-bundle":                        "dev-master",
"stof/doctrine-extensions-bundle":              "dev-master",
"heartsentwined/yuicompressor":                 "dev-master",
"jms/serializer-bundle":                        "dev-master",
"ali/datatable":                                "dev-master"
```

Bilaga 3. Flödesschema för klientens logik



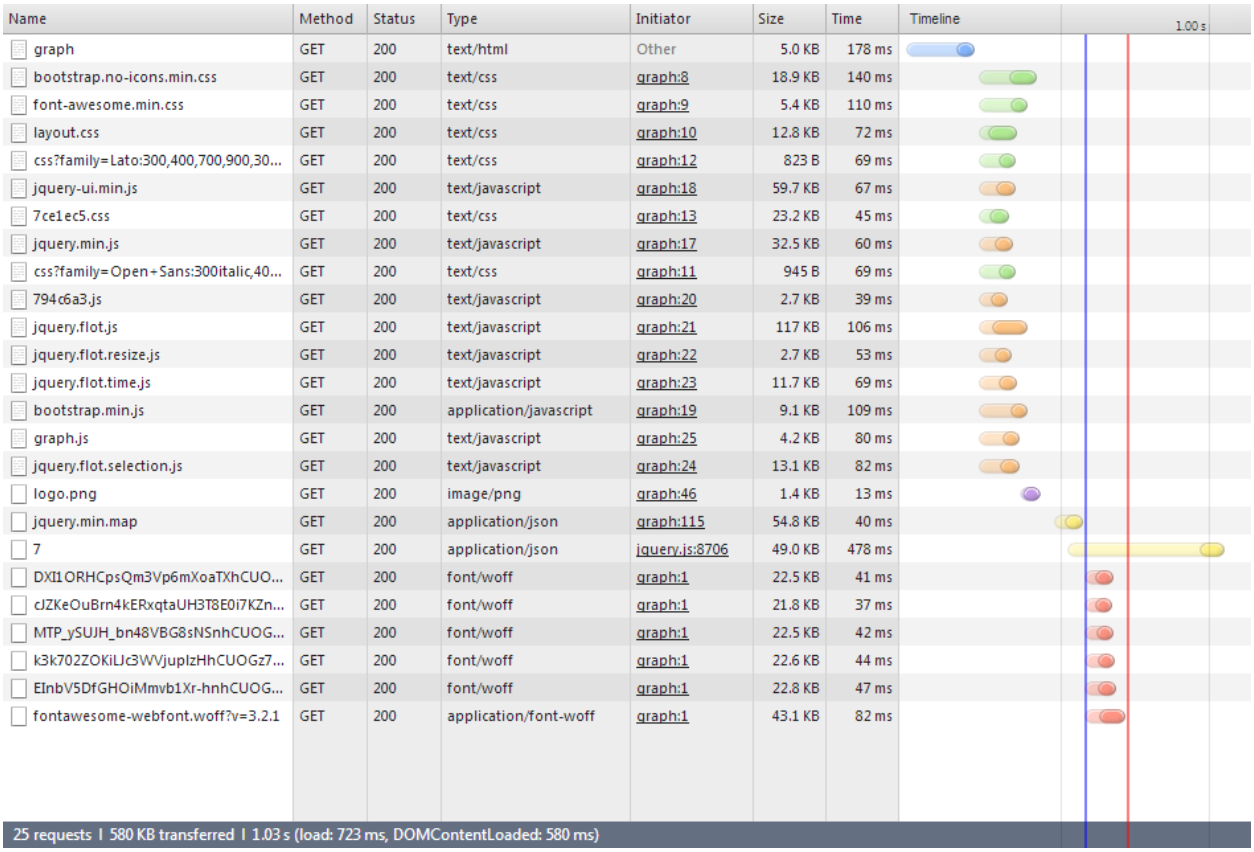
Bilaga 4. Laddning av sida utan mellanlagring i webbläsaren



Name	Method	Status	Type	Initiator	Size	Time	Timeline
graph	GET	200	text/html	Other	5.0 KB	138 ms	
bootstrap.no-icons.min.css	GET	200	text/css	graph:8	(from c...	Pending	
font-awesome.min.css	GET	200	text/css	graph:9	(from c...	1 ms	
layout.css	GET	200	text/css	graph:10	(from c...	Pending	
css?family=Open+Sans:300italic,40...	GET	200	text/css	graph:11	(from c...	Pending	
css?family=Lato:300,400,700,900,30...	GET	200	text/css	graph:12	(from c...	Pending	
7ce1ec5.css	GET	200	text/css	graph:13	(from c...	Pending	
jquery.min.js	GET	200	text/javascript	graph:17	(from c...	Pending	
jquery-ui.min.js	GET	200	text/javascript	graph:18	(from c...	Pending	
bootstrap.min.js	GET	200	application/javascript	graph:19	(from c...	Pending	
794c6a3.js	GET	200	text/javascript	graph:20	(from c...	Pending	
jquery.flot.js	GET	200	text/javascript	graph:21	(from c...	Pending	
jquery.flot.resize.js	GET	200	text/javascript	graph:22	(from c...	Pending	
jquery.flot.time.js	GET	200	text/javascript	graph:23	(from c...	Pending	
jquery.flot.selection.js	GET	200	text/javascript	graph:24	(from c...	Pending	
graph.js	GET	200	text/javascript	graph:25	(from c...	1 ms	
logo.png	GET	200	image/png	graph:46	(from c...	Pending	
7	GET	200	application/json	jquery.js:8706	49.0 KB	300 ms	
jquery.min.map	GET	200	application/json	Other	(from c...	3 ms	

19 requests | 54.0 KB transferred | 614 ms (load: 342 ms, DOMContentLoaded: 342 ms)

Bilaga 5. Laddning av sida med mellanlagring i webbläsaren



Name	Method	Status	Type	Initiator	Size	Time	Timeline
graph	GET	200	text/html	Other	5.0 KB	178 ms	
bootstrap.no-icons.min.css	GET	200	text/css	graph:8	18.9 KB	140 ms	
font-awesome.min.css	GET	200	text/css	graph:9	5.4 KB	110 ms	
layout.css	GET	200	text/css	graph:10	12.8 KB	72 ms	
css?family=Lato:300,400,700,900,30...	GET	200	text/css	graph:12	823 B	69 ms	
jquery-ui.min.js	GET	200	text/javascript	graph:18	59.7 KB	67 ms	
7ce1ec5.css	GET	200	text/css	graph:13	23.2 KB	45 ms	
jquery.min.js	GET	200	text/javascript	graph:17	32.5 KB	60 ms	
css?family=Open+Sans:300italic,40...	GET	200	text/css	graph:11	945 B	69 ms	
794c6a3.js	GET	200	text/javascript	graph:20	2.7 KB	39 ms	
jquery.flot.js	GET	200	text/javascript	graph:21	117 KB	106 ms	
jquery.flot.resize.js	GET	200	text/javascript	graph:22	2.7 KB	53 ms	
jquery.flot.time.js	GET	200	text/javascript	graph:23	11.7 KB	69 ms	
bootstrap.min.js	GET	200	application/javascript	graph:19	9.1 KB	109 ms	
graph.js	GET	200	text/javascript	graph:25	4.2 KB	80 ms	
jquery.flot.selection.js	GET	200	text/javascript	graph:24	13.1 KB	82 ms	
logo.png	GET	200	image/png	graph:46	1.4 KB	13 ms	
jquery.min.map	GET	200	application/json	graph:115	54.8 KB	40 ms	
7	GET	200	application/json	jquery.js:8706	49.0 KB	478 ms	
DXI1ORHCpsQm3Vp6mXoaTXhCUO...	GET	200	font/woff	graph:1	22.5 KB	41 ms	
cJZKeOuBrn4kERxqtaUH3T8E0i7KZn...	GET	200	font/woff	graph:1	21.8 KB	37 ms	
MTP_ySUJH_bn48VBG8sNSnhCUOG...	GET	200	font/woff	graph:1	22.5 KB	42 ms	
k3k702ZOKiLJc3WVjuplzHhCUOGz7...	GET	200	font/woff	graph:1	22.6 KB	44 ms	
ElnbV5DfGHOiMmvb1Xr-hnhCUOG...	GET	200	font/woff	graph:1	22.8 KB	47 ms	
fontawesome-webfont.woff?v=3.2.1	GET	200	application/font-woff	graph:1	43.1 KB	82 ms	

25 requests | 580 KB transferred | 1.03 s (load: 723 ms, DOMContentLoaded: 580 ms)